cursus cobol

2e herziene uitgave
inclusief structureel programma ontwerp

andrew parkin

ACADEMIC SERVICE

mobach
den akker 8
4054 md echteld
tel. 03440 - 1 78 56

CURSUS COBOL

2° herziene uitgave
inclusief structureel
programma ontwerp

cursus cobol

2e herziene uitgave
inclusief structureel programma ontwerp

andrew parkin

ACADEMIC
SERVICE

Oorspronkelijke titel: *COBOL for students, second edition, including structured program design*
Verschenen bij: Edward Arnold, Londcn, 1975
reprinted 1978, 1979, 1980
Second edition 1982

Vertaling: drs. A.M. Korff en drs. H.J. Stomps

CIP-GEGEVENS KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Parkin, Andrew

Cursus COBOL / Andrew Parkin ; [vert uit het Engels door A.M. Korff ... et al.]. - Den Haag : Academic Service. - Ill.
Vert. van: COBOL for students. - Londen : Arnold, 1982. - 2e dr. -
Met index. - Met structureel programma ontwerp.
ISBN 90-6233-095-9
SISO 365.3 SVS 8.12.3 UDC 681.3.06
Trefw.: programmeren ; computers.

eerste druk: 1978, bijdruk in 1980
tweede, herziene uitgave: 1983, bijdruk in 1984, 1986

Uitgegeven door: Academic Service
Postbus 81
2870 AB Schoonhoven
Druk: Krips Repro Meppel
Bindwerk: Meeuwis, Amsterdam
Omslagontwerp: JAM Gauw
ISBN 90 6233 095 9

VOORWOORD

AAN DE DOCENT

Ik denk dat de docent COBOL zichzelf de volgende belangrijke vragen stelt:

1. Doceer ik de Divisions in de volgorde waarin ze worden gecodeerd of in de volgorde van hun belangrijkheid voor de programmeringstaak? De eerste methode is recht door zee, maar de bijzonderheden van de Environment en Data Divisions kunnen in het gunstigste geval zonder betekenis zijn voor de student totdat hij iets geleerd heeft over de Procedure Division, en in het ongunstigste geval stomvervelend zijn. Doceren we met de alternatieve methode dan lopen we misschien de kans dat de student verloren raakt doordat hij geen zicht op een totaalbeeld krijgt.

2. Zet ik alle puntjes op de i gedurende de lessen, of concentreer ik me op de hoofdlijnen om de studenten in staat te stellen snel een eenvoudig programma te laten schrijven? Het eerste kan saai zijn of kan ertoe leiden dat de student de belangrijke kern in de lawine van details uit het oog verliest. Het tweede is aantrekkelijk, maar houdt in dat òf de student de leergang afsluit zonder de bijzonderheden te kennen, òf de docent moet teruggaan en dezelfde stof met meer oog voor het detail behandelen.

3. Doceer ik het COBOL-dialect dat geldt voor de computer die ik gebruik, of probeer ik het bijzondere te onderscheiden van het algemene? Het eerste is duidelijk een praktischer gerichte aanpak op korte termijn, maar het tweede kan waardevoller voor de student blijken op lange termijn.

4. Hoe doceer ik aan een groep met gemengde capaciteiten? Als de docent te snel gaat, verliest hij de langzamere en minder ervaren studenten, of hij zal op een verkeerd begrepen basiskennis voortbouwen. Als de docent te langzaam gaat en veel tijd besteedt aan het voorkomen van verkeerd begrip, riskeert hij dat de snellere student zijn aandacht geheel van het onderwerp afwendt.

5. Moet ik de cursus beginnen met het behandelen van programma structuren om daarna te laten zien hoe deze in COBOL weergegeven kunnen worden of breng ik eerst een basis van elementair COBOL bij om aan de hand daarvan het structureel programmeren te introduceren.

Ieder van deze problemen vormt een dilemma en daarom is er aanleiding voor een bespreking van het antwoord. Ik geef in het hierna volgende mijn redenen voor de keuze van de aldaar ontwikkelde oplossingen, maar ik hoop dat u dit boek zo flexibel vindt dat u de methode die u persoonlijk verkiest (bijvoorbeeld de stof in een andere volgorde behandelen) kunt uitvoeren.

DE AANPAK VAN DIT BOEK

Het boek bestaat uit drie delen. Deel 1 concentreert zich op het wezenlijke van de taal en brengt de student op een niveau waar hij een naar omvang bescheiden COBOL-programma dat gebruik maakt van seriegewijze bestanden kan schrijven. Deel 2 veronderstelt een zekere bekwaamheid in de grondbeginselen van COBOL en licht het ontwerpen van programma's nader toe en andere programmeringsvaardigheden die deel behoren uit te maken van het repertoire van de professionele programmeur. Deel 3 verbreedt de taalkennis van de student door een nadere toelichting van de meer geavanceerde kenmerken en mogelijkheden van COBOL.

Ieder deel bestaat uit een aantal hoofdstukken en elk hoofdstuk zou een leidraad voor één college kunnen zijn. De docent licht de stof in het hoofdstuk toe (hetgeen ongeveer 1 uur in beslag neemt), waarbij hij zich concentreert op de kerngedachten en zoveel details als gewenst is voor zijn doel overslaat. Vervolgens nodigt hij de studenten uit om de tekst van dat hoofdstuk te bestuderen.

Een hoofdstuk bestaat uit een aantal pregnant geschreven paragrafen gevolgd door vragen die een werkelijkheidsgetrouwe test van het begrip van de student, zowel ten aanzien van de kern als het detail, beogen te zijn. Snelle of ervaren studenten zijn in staat het hoofdstuk in ca. 1 uur af te maken, en veel studenten kunnen zo'n hoofdstuk zonder verdere hulp volledig in zich opnemen. De docent kan zijn aandacht dan concentreren op die studenten die het hoofdstuk niet uit zichzelf kunnen bevatten.

Om terug te keren tot de vijf vragen, dit boek beantwoordt ze als volgt:

1. Na het inleidende hoofdstuk (dat kan wegvallen voor ervaren studenten) wordt een beknopt overzicht gegeven van alle vier divisions. Daarna worden beschrijvingen uit de Procedure Division

en Data Division geïntroduceerd in de volgorde die ik voor een goed begrip natuurlijk vind. De student wordt op diverse plaatsen herinnerd aan de betrekkingen tussen de divisions. Voor de praktijkopgave aan het slot van hoofdstuk F geeft de docent aan de student over te nemen beschrijvingen van de Identification Division en Environment Division. Deze laatste divisions worden behandeld in het laatste hoofdstuk van deel 1 (hoofdstuk J) evenals een lijvige opgave gebaseerd op een programma dat gegevens valideert.

2. De docent kan zich op hoofdlijnen concentreren in zijn betoog. De student leert het detail uit het boek.

3. Het boek is onafhankelijk van een bepaalde computer en volgt de American National Standard COBOL. Waar de standaard ruimte laat voor interpretatie van de computerfabrikant, worden gebruikelijke variaties behandeld.

Waar ik een tegenstelling heb aangetroffen tussen een implementatie en de standaard, heb ik de variatie vermeld.

Slechts de algemeen gebruikelijke beschrijvingen in de Environment Division zijn behandeld. Het ingaan op alle finesses van USAGE COMPUTATIONAL en zijn variaties zou een enorm stuk werk betekenen. De docent oordele zelf over punten als deze, want hij kent de machine die de studenten gebruiken.

4. Omdat alle studenten de tekst van het vorige hoofdstuk hebben doorgewerkt, staan ze op een gelijkwaardiger ontwikkelingsniveau bij het begin van een les dan wellicht het geval is bij andere methodes. Vele algemeen verbreide misvattingen worden in het boek behandeld. Bovendien kan de docent, terwijl de tekst wordt doorgewerkt, zijn aandacht concentreren op het helpen van die studenten die daaraan het meest behoefte hebben.

5. Over het algemeen kan de toelichting op gestructureerd programmeren nauwkeuriger en zinvoller zijn, als de student geacht wordt te beschikken over een reeds bestaande kennis m.b.t. programmeren en de moeilijkheden van probleemoplossing met de in de taal beschikbare besturingsstructuren heeft ervaren. Daarom wordt het ontwerpen van programma's naar Deel 2 verschoven. Indien de student evenwel COBOL als tweede taal bestudeert, is er een hele goede reden om te beginnen met een aantal onderwerpen uit Deel 2.

AAN DE STUDENT

Heeft u het juiste boek in handen?

Dit is geen boek voor zelfstudie. Het is bedoeld om een serie lessen te begeleiden.

Anderzijds kunt u, als u al enige notie heeft van het programmeren van een computer, dit boek beschouwen als een zeer snelle manier om COBOL meester te worden. Als u al een beetje COBOL kent, kunnen de delen 2 en 3 van het boek uw horizon verbreden.

Hoe COBOL te leren?

Wanneer u een vreemde taal - bijvoorbeeld Engels - leert, dan kunt u de theorie en structuur ervan begrijpen door te luisteren naar een docent of een studieboek erover te lezen. Dit zal u niet veel helpen wanneer u van de boot in Harwich stapt. De enige weg naar welbespraaktheid in de taal is het spreken van die taal. Het verbeteren van de uitspraak van de woorden, het vormen van uw eigen bewoordingen en zinnen zijn wezenlijke bestanddelen van het leren en zich herinneren.

Het leren van COBOL gaat evenzo, zij het dat COBOL alleen als een geschreven taal bestaat. Vaardigheid in COBOL wordt verworven door het schrijven ervan.

Dit boek bevat vele opgaven die vragen om geschreven antwoorden; het is van groot belang dat u de antwoorden opschrijft. Geestdriftige studenten worden soms verleid om de vragen 'in hun hoofd' te beantwoorden, ongeduldig als ze zijn om vorderingen te maken met het onderwerp. Biedt weerstand aan deze verleiding, want anders kan het er wel eens op uitlopen dat u in Harwich aankomt en alleen maar gelezen heeft over Engels.

Hoe dit boek te gebruiken?

Ieder hoofdstuk van het boek (hoofdstuk A, hoofdstuk B enzovoort) omvat een aantal paragrafen (paragraaf A1, paragraaf A2 enzovoort). Aan het eind van iedere paragraaf staan enige vragen. Lees de paragraaf en schrijf de antwoorden bij de vragen op.

Als u een vraag niet kunt beantwoorden, herlees dan de belangrijke paragraafgedeeltes. Als u het spoor bijster raakt, of als er iets niet duidelijk is, vraag het dan aan uw docent.

DANKWOORD

Opnieuw moet dank worden betuigd aan Roger Barnes, David Howe en Derek Ruffhead voor hun assistentie bij de eerste druk. Er is nauwelijks een dag voorbijgegaan sinds de publicatie van de eerste druk zonder dat ik de onwaardige ontvanger was van een waardevolle gedachte mij aangedragen in het leslokaal, de docentenkamer of via de post. Zelfs al zou ik me al deze bronnen kunnen herinneren, de lijst met dankwoorden zou te lang worden om ze allemaal te omvatten. Mijn waardering gaat uit naar de vele

studenten en de staf van Leicester Polytechnic en het New South Wales Institute of Technology voor hun hulp evenals naar de Nederlandse vertalers van de eerste druk, H.J. Stomps en A.M. Korff die met hulp van hun studenten een goudmijn aan correcties en ideeën aandroegen. Bovendien ben ik bijzondere dank verschuldigd aan Bob Coats voor zijn door hem gegeven tijd en raad.

Een woord van dank is ook verschuldigd aan die organisaties die hebben bijgedragen aan de ontwikkeling van COBOL. Ik geef hier een mededeling op hun verzoek weer:

INHOUD

DEEL 1

Basisbegrippen

A GRONDSLAGEN EN BEGRIPPEN - EEN OVERZICHT

A1 DE COMPUTER



FIGUUR 1 Algemeen overzicht van een computer (IBM-foto)
1) magneetbandeenheden 2) ponskaartlezer 3) beeldbuisterminal
4) behuizing CVE en geheugen 5) operator's console 6) printers
7) magneetschijfgeheugens.

Een digitale computer bestaat uit een *centrale verwerkingseenheid* (CVE), waarop via kabels bepaalde *randapparaten* zijn aangesloten.

De CVE heeft drie hoofdonderdelen: een *besturingsorgaan*, dat alle werkzaamheden van de machine bestuurt en de opdrachten vervat in het programma uitvoert; een *rekenorgaan*, dat optelt,

vermenigvuldigt, enzovoort; en een *geheugen*, dat het uit te voeren programma opslaat evenals de gegevens waarmee het lopende programma werkt.

Het besturingsorgaan haalt een opdracht uit het geheugen, voert deze uit, haalt de volgende opdracht, voert deze uit, enzovoort.

Alle programma's beginnen met enige oorspronkelijke gegevens (*invoer*) die worden verwerkt om nieuwe gegevens (*uitvoer*) te produceren.

De machine voert slaafs de opdrachten uit en het is voor een computer feitelijk onbestaanbaar een opdracht foutief uit te voeren. Als de uitvoer niet klopt, dan is er dus bijna zeker een fout in de invoergegevens geslopen of een logische fout in de programma-opdrachten.

De randapparaten zijn afzonderlijk opgestelde machines, die bestuurd worden door de CVE. Er bestaat *invoerapparatuur* (voor de verstrekking van gegevens en programma's aan het geheugen), *uitvoerapparatuur* (voor het uitvoeren van de programmaresultaten vanuit het geheugen) en *opslagapparatuur* (die hoofdzakelijk gebruikt wordt voor de opslag van de uitvoergegevens die in een later stadium opnieuw zullen moeten worden ingevoerd voor verdere verwerking).

Opgaven

Als u een vraag niet direct kunt beantwoorden, probeer dan eerst het antwoord in de paragraaf te vinden. Toets pas daarna uw antwoorden aan de oplossingen aan het eind van het hoofdstuk. Het maken van de opgaven wordt voor het verwerken van de stof ten zeerste aanbevolen.

1. Welk doel heeft een invoerapparaat?
2. Welk doel heeft opslagapparatuur?
3. Welke twee hoofdoorzaken van fouten in de uitvoer van de computer bestaan er?
4. Waar is het computerprogramma tijdens zijn verwerking opgeslagen?
5. Kunnen uitvoergegevens ook als invoer gebruikt worden?

A2 BEELDSCHERMTERMINAL

Een van de gebruikelijkste invoerapparaten is het aan een beeldscherm gekoppelde toetsenbord, ook wel algemeen een terminal genaamd (zie figuur 2).



FIGUUR 2 Een beeldschermterminal met toetsenbord

Het beeldscherm van de meeste terminals kan 20 regels met 80 tekens per regel zichtbaar maken. Naar de positie van een teken in een regel wordt dikwijls verwezen d.m.v. zijn *kolomnummer*. Zo staat het eerste teken in een regel in kolom 1; het twintigste teken in een regel zou dan staan in kolom 20.

Ieder opvolgend teken dat wordt aangeslagen op het toetsenbord wordt gewoonlijk afgebeeld op het scherm. De aangeslagen tekens worden dikwijls opgeslagen in een *buffergeheugen* binnen de beeldschermterminal, totdat de RETURN-toets wordt ingedrukt. Wanneer dit gebeurt, worden alle tekens in de bewuste regel overgebracht naar de centrale verwerkingseenheid. Zo is de basiseenheid waarmee de operator krijgt te maken de regel met tekens; de operator maakt een regel klaar en drukt, wanneer het zover is, op de RETURN-toets om de regel aan de computer over te dragen.

Indien de terminal niet over een buffergeheugen beschikt, wordt ieder afzonderlijk teken naar de computer gezonden, zodra

de toets is ingedrukt. Terminals zonder buffergeheugen zijn tamelijk gebruikelijk in het bedrijfsleven, maar aan computers gekoppelde terminals ten behoeve van instructie aan programmeurs zijn dikwijls van het type met een buffergeheugen.

Figuur 2 illustreert eveneens een ander invoerapparaat dat soms wordt gebruikt bij terminals zonder buffergeheugen: de *licht-pen*. Dit apparaat tast het licht, dat van het scherm komt, af en brengt vervolgens aan de CVE de coördinaten over van het punt op het scherm, waarop de pen is gericht.

De terminal kan ook worden gebruikt als uitvoerapparaat. Een in het geheugen van de computer gemaakte boodschap wordt naar de terminal gezonden om te worden getoond op het scherm. Gewoonlijk wordt slechts één regel tegelijk verzonden, maar op een terminal met 20 regels blijven de vorige 19 regels zichtbaar, tenzij de instructie is gegeven het scherm leeg te maken. De vertoning (van de boodschap) gaat verloren, wanneer de terminal wordt uitgeschakeld. Wanneer er een permanente boodschap wordt vereist, verdient het wellicht de voorkeur een 'harde kopie' af te drukken, zoals wordt uitgelegd in de volgende paragraaf.

De programmeur dient een beslissing te nemen waar ergens op de regel een gegeven moet worden ingetoetst. De kolommen die zijn toegewezen voor het intoetsen van een bepaalde groep tekens worden een *veld* genoemd. De regel is dus verdeeld in velden, die ieder weer één of meer tekens bevatten. Soms is het wenselijk te bedenken dat een veld is verdeeld in twee of meer kleinere velden; in dit geval wordt het grotere veld een *groepsveld* genoemd. Niet verder onderverdeelde velden heten *elementaire* velden.

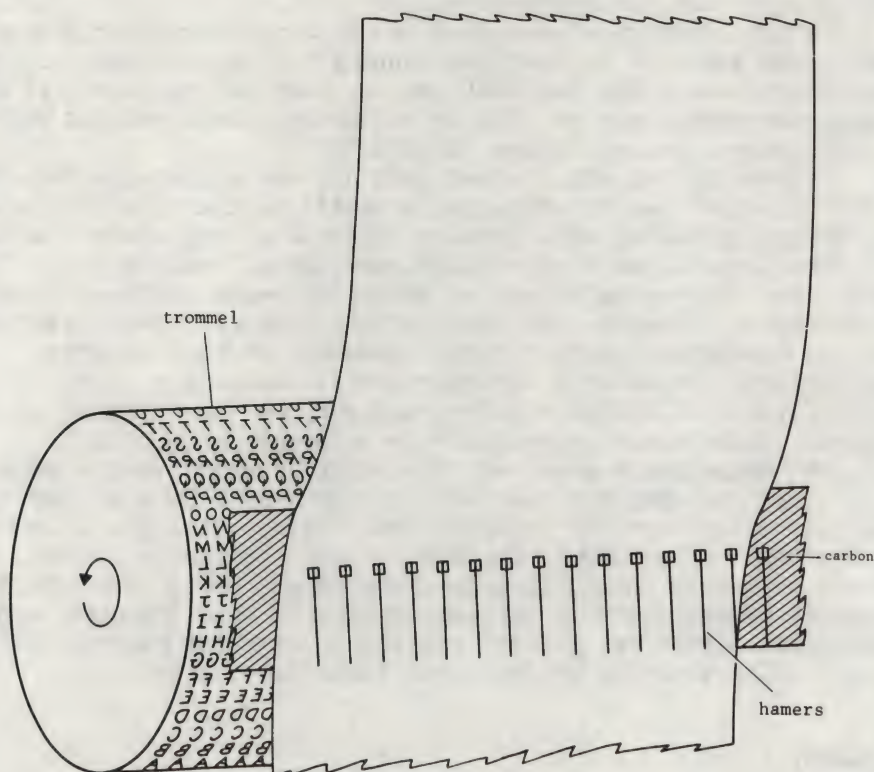
Opgaven

1. Hoeveel kolommen heeft de beeldschermterminal gewoonlijk? Hoeveel tekens kunnen gewoonlijk op één regel worden ingevoerd? Hoeveel tekens kunnen op het gehele scherm worden afgebeeld?

2. Als een regel een naam (13 tekens) bevat, een adres (40 tekens) en een rekeningnummer (6 cijfers), hoeveel kolommen worden dan gebruikt? Hoeveel elementaire velden zijn er in die regel?

3. Het veld 'rekeningnummer' in een regel bevat een veld provincie (1 cijfer), een veld klantensoort (1 cijfer) en een veld volgnummer (4 cijfers). Hoe groot is het groepsveld?

A3 DE REGELDRUKKKER



FIGUUR 3 Regeldrukker met trommel

Het gebruikelijkste middel voor de uitvoer van voor de mens leesbare gegevens is de regeldrukker. Het belangrijkste type regeldrukker (figuur 3) heeft een *trommel*, maar er bestaan ook andere, bijvoorbeeld kettingdrukkers, matrixdrukkers, letterwieldrukkers, inktstraaldrukkers en laserdrukkers.

De metalen omtrek van de trommel is gegraveerd met alle tekens - een rij A's, een rij B's, enzovoort. Tegenover elk teken van zo'n rij bevindt zich een *hamertje*. Het papier en het karbon worden tussen de hamertjes en de trommel ingeschoven.

Stel nu dat in het geheugen een gehele regel klaar staat om te worden afgedrukt. De regeldrukker wordt aangezet en de trommel begint met grote snelheid rond te draaien. Zodra een rij tekens tegenover de hamertjes verschijnt, slaan die hamertjes aan, die overeenkomen met de posities in de regel waar dat teken gedrukt moet worden. Daarbij drukken de hamertjes het papier op

het teken met het karbon daartussenin. Op deze wijze worden alle A's in een regel afgedrukt, daarna alle B's, de C's, enzovoort, totdat de gehele regel is gevuld.

Het papier zelf is een waaiervormig, in pagina's van gewoonlijk 66 regels verdeeld, *kettingformulier*. Het wordt op zijn plaats gehouden en door de regeldrukker gevoerd met behulp van tandraderen die grijpen in gaten aan beide kanten van het papier.

Een regeldrukker is een machinerie van de hoogste precisie en bereikt snelheden van 300 tot 2000 regels per minuut.

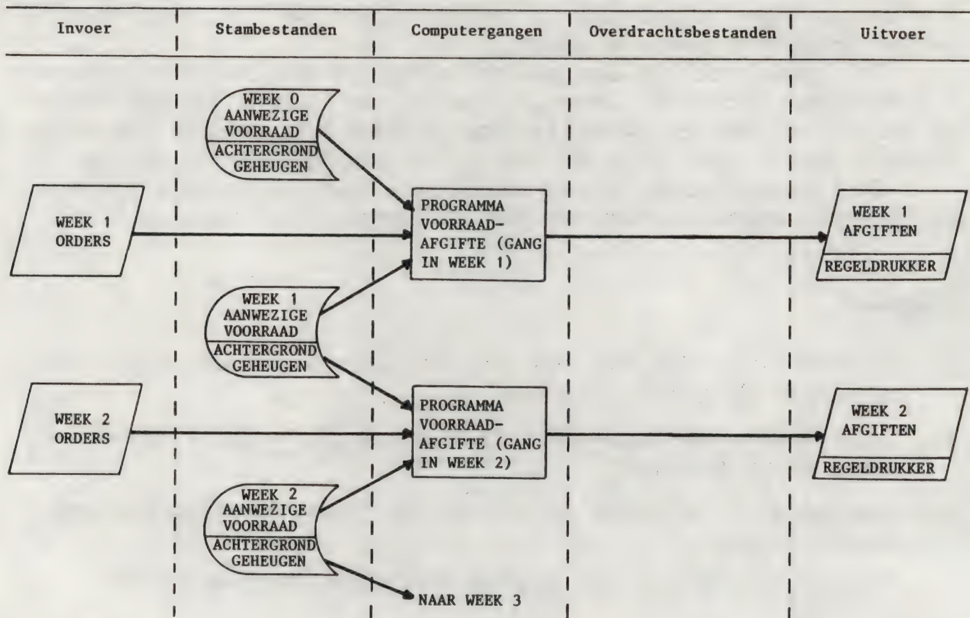
Opgaven

1. Hoeveel hamertjes zou een regeldrukker tellen die regels met maximaal 132 tekens afdrukt?
2. Hoeveel omwentelingen van de trommel zijn vereist om een regel af te drukken?
3. Hoe wordt de aanvoer en het op z'n plaats houden van het papier geregeld?
4. Hoeveel regels kan een pagina afdrukpapier gewoonlijk bevatten?
5. Als een regeldrukker 600 regels per minuut afdrukt, in hoeveel tijd wordt dan één regel afgedrukt?

A4 OPSLAGAPPARATUUR

Niet alle programmaresultaten behoeven te worden afgedrukt. Veronderstel bijvoorbeeld dat een programma een lijst van afgegeven goederen van een warenhuis moet opleveren, waarbij de orders die per week geplaatst zijn als invoer dienen. Het is tevens gewenst dat daarbij de voorraad van alle in omloop zijnde goederen wordt bijgehouden in een vorm die de computer kan lezen. De afgegeven goederen zullen de omvang van deze voorraad doen verminderen, hetgeen resulteert in een *bijgewerkte* voorraad per soort goederen. Het is niet noodzakelijk deze bijgewerkte voorraden af te drukken; ze kunnen uitgeschreven worden naar een *achtergrondgeheugen* en het aldus gecreëerde *bestand* kan de week daarop bij het *mutatie*programma van die week wederom ingevoerd worden (zie figuur 4).

De belangrijkste vormen van achtergrondgeheugen zijn momenteel *magneetbanden* en *magneetschijven*. De informatie wordt door een *magneetbandstation* op de magneetband vastgelegd. Dit apparaat werkt op dezelfde manier als een bandrecorder thuis (of



FIGUUR 4 Het bijwerken van een bestand

als een cassetterecorder ingeval van microcomputers), maar hij doet dat wel veel sneller en de magneetband is ook veel langer (ca. 730 meter is een gebruikelijke maat). Aangezien gegevens op de band vastgelegd kunnen worden met een dichtheid van zo'n 630 tekens per cm, kunnen verscheidene miljoenen tekens op één enkele band worden opgenomen. Dit is veel meer informatie dan in het geheugen van de CVE kan worden opgeslagen, vandaar dat programma's worden geschreven waarbij slechts één *record* tegelijk wordt ingelezen en verwerkt en mogelijke resultaten worden uitgevoerd, om vervolgens het volgende record in te kunnen lezen, te verwerken, enzovoort.

Record is de term voor een verzameling bij elkaar behorende velden. Zo zouden de gegevens op een regel van een beeldscherm een record kunnen vormen; de gegevens in een afgedrukte regel eveneens. Voorbeelden van bestanden en records op magneetband zijn:

Artikelbestand - ieder record in het bestand zou kunnen bevatten:

1. een artikelnummer
2. een artikelbeschrijving
3. de voorraad

enzovoort, enzovoort.

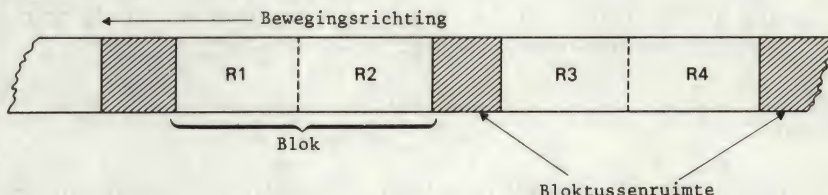
Werknemersbestand (voor salarisadministratie) - een record zou kunnen bevatten:

1. werknemersnummer
2. Werknemersnaam
3. Maandsalaris
4. inhoudingen
5. salaris tot op heden
6. belasting tot op heden

enzovoort, enzovoort.

Ik schrijf 'enzovoort, enzovoort' omdat in de praktijk de records in dergelijke bestanden veel meer velden hebben dan de nu opgesomde. Maar de aard van die extra velden is afhankelijk van het specifieke gebruik van de bestanden.

Informatie wordt vastgelegd op magneetband in *blokken* van gegevens. Het vastleggen kan namelijk alleen plaatsvinden wanneer de band zich met volle snelheid beweegt. Omdat de band iedere keer nadat een blok is gekopieerd moet stoppen, wordt na ieder blok een stukje van de band opengelaten voor het tot stilstand en weer op volle snelheid komen van de band (zie figuur 5).



FIGUUR 5 Blokken en records op magneetband

Deze tussenruimte is in werkelijkheid ongeveer 1,3 cm lang. Als nu het aantal tekens in de blokken gering is (bijvoorbeeld 80) dan zal de magneetband een grote hoeveelheid ruimtes tussen de blokken en verhoudingsgewijs erg weinig gegevens bevatten. Om deze reden en ook om de frequentie van het stoppen te verminderen zouden wij de blokken zo groot mogelijk willen maken; de beperkende factor is echter de hoeveelheid reserveruimte in het geheugen die nodig is om een blok in te lezen of uit te schrijven. Het is mogelijk dat de lengte van een record veel kleiner is dan de gewenste bloklengte. Het is dan ook gewoonlijk noodzakelijk twee of meer records samen te groeperen om een bruikbare bloklengte te krijgen (zoals bij R1 en R2 in figuur 5). Gelukkig bestaan er in COBOL faciliteiten voor het automatisch verwerken van records tot blokken.

Wanneer we een bestand voor een COBOL programma volledig willen specificeren, dan moeten we dat op vier niveaus doen:

NIVEAU	GESPECIFICEERD WORDT:
BESTAND	- de naam van het bestand en het apparaat waarop het is (wordt) vastgelegd
BLOK	- het aantal records in een blok
RECORD	- de velden die een record vormen
VELD	- de tekens die het veld vormen.

Het eerste record op een magneetband heeft een bijzonder karakter: de *koplabel*. Deze identificeert het bestand. Er bestaat ook een *staartlabel* die het einde van het bestand aanduidt.

In wezen zijn dezelfde overwegingen van toepassing op bestanden op magneetschijven, behalve dan dat de schijven ook directe toegankelijkheid tot een benoemd record mogelijk te maken. In figuur 5 moet de computer bijvoorbeeld om record 4 in te lezen op normale magneetband doorzoeken tot na de records 1, 2 en 3. Maar bij magneetschijven kan record 4 direct worden teruggevonden. Hoe dit in zijn werk gaat, wordt verklaard in hoofdstuk T.

Een moderne commerciële bandeenheid kan evenveel tekens lezen als in dit gehele boek staan in een tijd die ligt tussen de 10 seconden en twee minuten, afhankelijk natuurlijk van de snelheid waarmee de band beweegt en de tijd die het vergt om te stoppen en opnieuw te beginnen in de bloktussenruimtes, evenals van het aantal in een blok gekozen tekens.

Opgaven

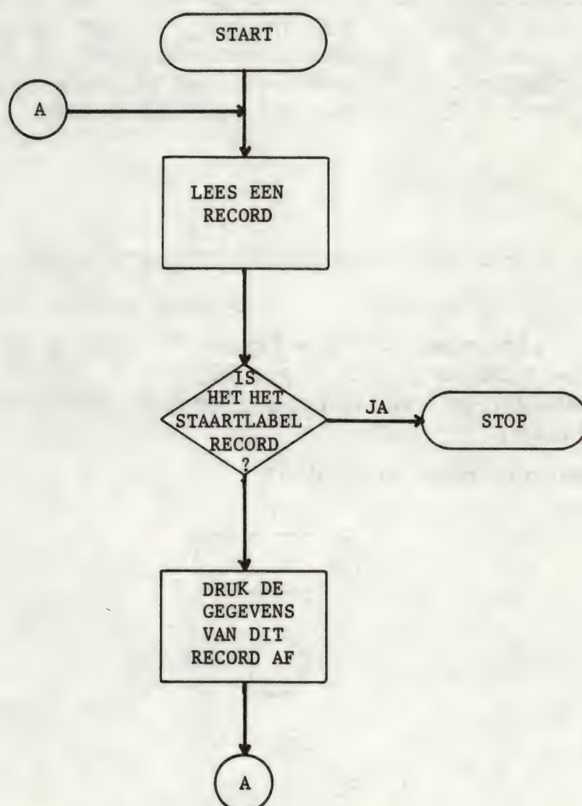
1. Noem vier typen randapparatuur. Classificeer ze als invoer-, uitvoer- of opslagapparaten.
2. Waarvoor dient een koplabel?
3. Hoe worden blokken gegevens gescheiden op magneetband?
4. Wat beperkt in het algemeen de lengte van een blok?
5. Welke vier niveaus van bestandsbeschrijving worden vereist door een COBOL-programma?
6. Bevat een salarisprogramma de naam van iedere werknemer in de organisatie? Zo niet, waar worden dan de namen bewaard?

A5 DE PROGRAMMASPECIFICATIE

Voordat een COBOL-programma wordt geschreven, moet een tot in onderdelen uitgewerkte specificatie ontworpen worden die precies aangeeft wat het programma moet doen. Deze bestaat uit drie hoofdonderdelen:

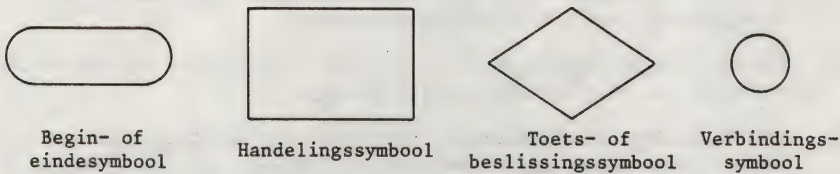
1. Invoer: een beschrijving van de bestands- en recordstructuren van de invoergegevens
2. Uitvoer: een beschrijving van de bestands- en recordstructuren van de uitvoergegevens
3. Verwerking: een beschrijving van de procedures die het programma moet uitvoeren om op basis van de invoer de uitvoer op te leveren.

Het verwerkingsgedeelte van de specificatie wordt gewoonlijk door de programmeur weergegeven in een *stroomschema*. Dit is een schema dat de logische volgorde van vragen die beantwoord en opdrachten die uitgevoerd moeten worden, aangeeft. Bijvoorbeeld:



FIGUUR 6 Stroomschema: het afdrukken van gegevens die op een magneetbandbestand staan

Dit programma zal de gegevens die op het magneetbandbestand staan op de regeldrukker afdrukken. De gebruikte symbolen zijn:

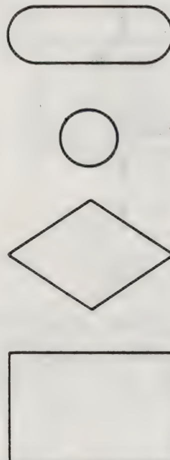


FIGUUR 7 Stroomschema symbolen

Het stroomschema is een basismethode voor het beschrijven van van processen en het wordt dan ook wijd en zijd gebruikt in vele industrietakken. Maar heden ten dage wordt wel ingezien, dat het werken met stroomschema's een foutengevoelige aanpak is bij grote computerprogramma's. Een betere methode wordt in Deel 2 van dit boek geïntroduceerd. Intussen dient het gebruik van stroomschema's beperkt te blijven tot eenvoudige gevallen.

Opgaven

1. Noem de drie onderdelen van een programmaspecificatie.
2. Hoe kan de logica van een programma worden weergegeven?
3. Breid het stroomschema van figuur 6 zodanig uit dat tevens het aantal records wordt opgeteld en het resultaat afgedrukt. Dit resultaat moet pas worden afgedrukt nadat alle records zijn ingelezen.
4. Wat betekenen deze symbolen?



FIGUUR 8 Opgave 4

A6 DE COBOL-VERTALER

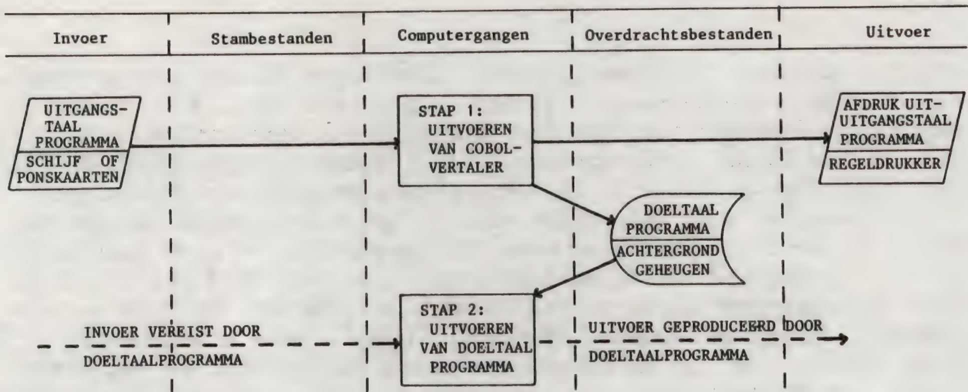
COBOL (Common Business Oriented Language) is een computertaal, die in 1959 in de Verenigde Staten ontwikkeld werd. Het verantwoordelijke lichaam voor de handhaving en verdere ontwikkeling van de taal is de Conference on Data Systems Language (CODASYL), een vrijwillige organisatie die vertegenwoordigers van gebruikers en producenten van computers omvat. Het Amerikaanse Standards Institute stelde in 1966 voor een standaardversie van de taal in te voeren, gebaseerd op de aanbevelingen van de CODASYL uit 1965. Deze standaardversie werd bekrachtigd in 1968 en wordt sindsdien genoemd American National Standard COBOL (ANS COBOL) of soms COBOL '68. In 1974 werd een sterk uitgebreide en herziene versie van de taal aanvaard voor ANS COBOL en deze is nu overal beschikbaar. Dit boek is gebaseerd op deze versie, COBOL '74, ofschoon ook verwezen wordt naar talrijke voorgestelde uitbreidingen en verbeteringen van de taal tot 1982. Deze uitbreidingen zijn niet overal beschikbaar.

Omdat COBOL dicht bij onze spreektaal staat en niet is beperkt tot de faciliteiten van één bepaalde computer, wordt het een *hogere programmeertaal* genoemd (*lagere programmeertalen*, we spreken eigenlijk van *assembleertalen*, staan dichter bij de taal, de machinecode, die intern door de computer wordt gebruikt en zijn meestal wel beperkt tot die machine). Hogere programmeertalen worden soms *probleemgericht* genoemd, terwijl assembleertalen *machinegericht* zijn.

Gewoonlijk noteert de programmeur de opdrachten van zijn programma op *coderingsbladen*. Daarna wordt het programma gewoonlijk overgebracht op een magneetschijfbestand door het in te toetsen op een terminal. Het door de programmeur geschreven programma wordt het *uitgangstaalprogramma* genoemd.

Het uitgangstaalprogramma moet vertaald worden in de door de computer intern gebruikte taal. Deze taal wordt de *machinecode* genoemd en het in machinecode vertaalde programma wordt het *doeltaalprogramma* genoemd. Het vertalingsproces zelf wordt uitgevoerd door een computerprogramma: 'de COBOL-vertaler' (Eng.: compiler).

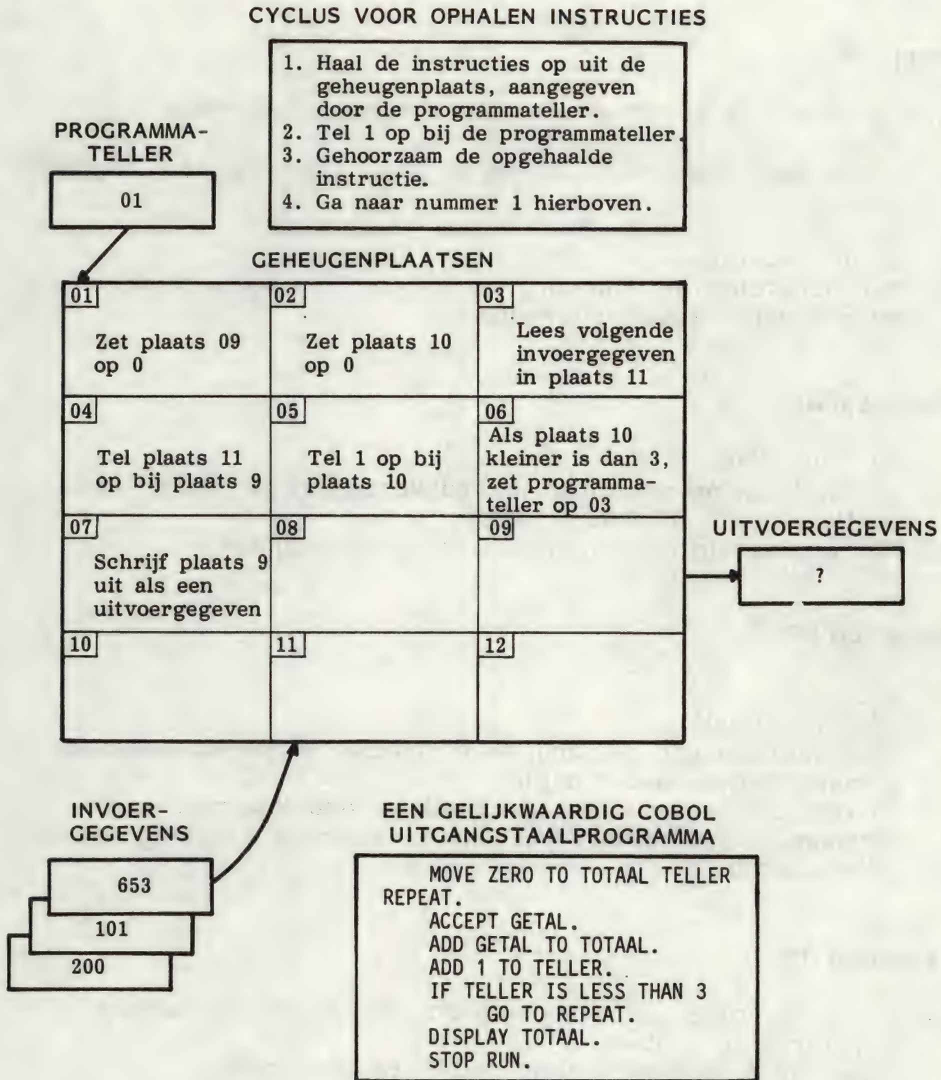
Nadat de computer het uitgangstaalprogramma van de schijf (of van ponskaarten) heeft ingelezen, vertaalt de COBOL-vertaler het dus in een doeltaalprogramma in machinecode. Is dit doeltaalprogramma daarbij naar het achtergrondgeheugen weggeschreven, dan kan het vervolgens in de machine geladen en uitgevoerd worden. Het proces kan geïllustreerd worden als in figuur 9, bladzijde 14.



FIGUUR 9 Het vertaalproces

Opgaven

1. Wat is een vertaler?
2. Hoe heet het programma dat de programmeur schrijft?
3. Welke naam wordt gegeven aan het programma nadat het is vertaald in machinecode?
4. Waar is COBOL een afkorting van?
5. Doe in figuur 10 alsof u de besturingseenheid bent die de cyclus voor het ophalen van instructies volgt. Wat is de uitvoer, wanneer instructie nr. 7 is volbracht?



FIGUUR 10 Hoe een computer werkt, in nader detail. Het uitgangstaalprogramma, geschreven door de programmeur, wordt door de computer vertaald in een doeltaalprogramma, in machinecode. Het doeltaalprogramma wordt opgeslagen in de geheugenplaatsen van de CVE. De besturingseenheid in de CVE heeft een interne logica die slechts de cyclus voor het ophalen van instructies uitvoert.

ANTWOORDEN HOOFDSTUK A

Paragraaf A1

1. Om gegevens en programma's in het computergeheugen in te lezen.
2. Om tussentijdse resultaten op te slaan die later weer ingelezen worden.
3. Fout in de gegevens; fout in het programma.
4. In het geheugen.
5. Ja; resultaten die zijn uitgevoerd naar een opslagapparaat kunnen later ingevoerd worden.

Paragraaf A2

1. 80; 80; 1600.
2. 59; 3. Merk op dat dit antwoord veronderstelt dat er geen spaties tussen de velden zijn.
3. Het groepsveld 'rekeningnummer' heeft 6 cijfers.

Paragraaf A3

1. 132.
2. Eén (maximaal).
3. Door tandraderen die grijpen in speciaal daarvoor bestemde transportgaten in het papier.
4. 66 (6 x 11); maar veel regeldrukkers drukken niet op de bovenste en onderste regel af, dus dan wordt het 64 regels.
5. 1/10 seconde.

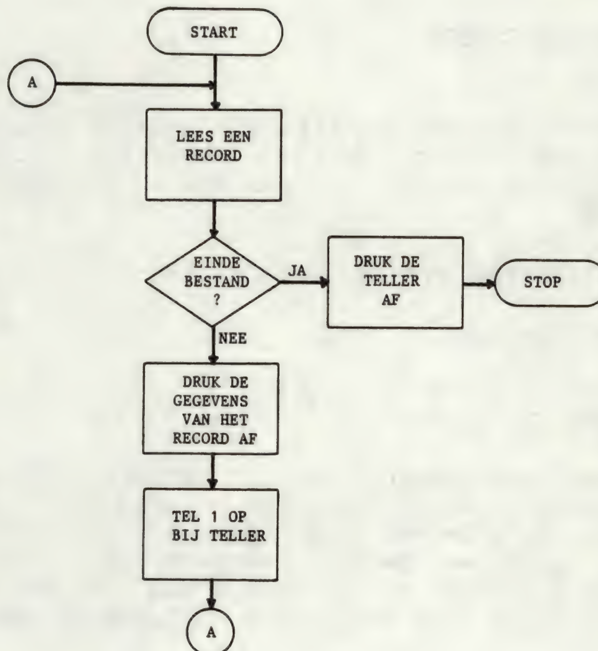
Paragraaf A4

1. Beeldschermterminal (toetsenbord-invoer: scherm-uitvoer).
   Regeldrukker - uitvoer.
   Magneetbandstation, magneetschijfstation - opslag.
2. Om het bestand te identificeren.
3. Door een tussenruimte tussen de blokken.
4. De hoeveelheid reserveruimte (d.w.z. de ruimte die niet door het programma of andere gegevens in beslag wordt genomen) die beschikbaar is in het geheugen om het blok in op te bouwen vóórdat het uitgeschreven wordt. (Bij computers met een groot geheugen kan de beperking van de lengte toch noodzakelijk zijn door eigenschappen van de opslagapparatuur.)
5. Bestand; blok; record; veld.
6. Neen. De namen van de werknemers (samen met andere gegevens, zoals hun maandsalaris en belastingkortingen)

worden in een bestand opgeslagen - waarschijnlijk op magneetband, één record per werknemer.

Paragraaf A5

1. Een beschrijving van de invoer, uitvoer en verwerking.
2. Door een programmastroomschema.
3. (Aangenomen dat de teller begint bij 0).



FIGUUR 11 Antwoord op opgave 3

4. Begin- of eindesymbool; verbindingssymbool; testsymbool; handelingssymbool.

Paragraaf A6

1. Een computerprogramma dat de in een hogere programmeertaal gegeven instructies vertaalt in de machinecode.
2. Uitgangstaalprogramma.
3. Doeltaalprogramma.
4. Common Business Oriented Language.
5. 954.

B HOOFDZAKEN VAN COBOL

B1 SYNTAXIS VAN COBOL

COBOL-opdrachten worden geschreven in *zinnen*. Een opdracht is een uitdrukking die begint met een COBOL-werkwoord. Een zin wordt beëindigd door een punt en kan één of meer opdrachten bevatten. Zo is

```
SUBTRACT BELASTING FROM SALARIS.
```

een zin met één opdracht, terwijl

```
SUBTRACT BELASTING FROM SALARIS
ADD BIJSLAG TO SALARIS.
```

een zin met twee opdrachten is. (In dit voorbeeld zouden BELASTING en BIJSLAG de namen kunnen voorstellen van drie velden, waarvan de gegevens in het geheugen zijn ingelezen. De opdrachten instrueren de computer de inhoud van het veld BELASTING af te trekken van de inhoud van het veld SALARIS en de inhoud van het veld BIJSLAG op te tellen bij die van het veld SALARIS.)

Zinnen kunnen gegroepeerd worden in *paragrafen*. Een paragraaf bevat één of meer zinnen. (Onder bepaalde omstandigheden is het gewenst een paragraaf zonder zinnen te hebben; dit zal in een ander hoofdstuk behandeld worden.)

Paragrafen kunnen op hun beurt gegroepeerd worden in *sections* (secties). Het COBOL-vertaalprogramma verwacht dat sommige paragrafen en sections in ieder COBOL-programma verschijnen. Het herkent ze aan de specifieke benamingen. De programmeur zal vaak besluiten nog andere paragrafen of sections in te voeren, waarvoor hij dan zelf namen bedenkt.

De zinnen, paragrafen en sections worden gegroepeerd in *divisions*. Er zijn slechts vier divisions die alle aanwezig moeten zijn in een volledig COBOL-programma. Het zijn:

```
IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
DATA DIVISION.
PROCEDURE DIVISION.
```

De divisions staan in bovengenoemde volgorde in een COBOL-programma.

Iedere division heeft een afzonderlijk doel, zoals uit het volgende blijkt:

IDENTIFICATION DIVISION: Bevat de naam van de programmeur, naam en doel van het programma en plaats en tijd waarop het werd geschreven. Dit zijn grotendeels commentaren die door de compiler genegeerd worden, afgezien van de controle op syntactische correctheid.

ENVIRONMENT DIVISION: Beschrijft het soort computer dat de uitgangstaal- en doeltaalprogramma's moet compileren, respectievelijk verwerken en andere details van de computerconfiguratie.

DATA DIVISION: Beschrijft alle gegevens en bestanden, die door het programma gebruikt worden. De aard van ieder bestand wordt in detail beschreven, evenals de structuur van de erin vervatte gegevenselementen. De compiler gebruikt deze informatie om een gebied in het geheugen te reserveren voor de blokken met gegevens die moeten worden ingelezen of uitgeschreven. Gegevens die geen deel uitmaken van een bestand worden beschreven in de *working-storage* section.

PROCEDURE DIVISION: Hier schrijft de programmeur opdrachten die de compiler vertellen welke bewerkingen moeten worden uitgevoerd.

Opgaven

1. Hoe weet de compiler waar een zin eindigt en een andere begint?
2. Bekijk de codering van het voorbeeld coderingsformulier in figuur 12. (De 77 die in dit programma verschijnt zal later toegelicht worden; hij wordt alleen gebruikt in de working-storage section en zegt iets over het type van het betreffende veld.) Als het programma uitgevoerd wordt, wat zou dan het resultaat zijn?
3. a) Hoeveel geheugenposities denkt u, dat de compiler zal reserveren voor de gegevens van het programma in figuur 12?
   b) Kunt u zeggen hoeveel geheugenposities nodig zijn voor de bewerkingen?
4. U mag de naam van een randapparaat niet in de Data Division noemen. In welke division kunt u de compiler de namen van de

Blad van bladen

Computer	Project	Programma	Omschrijving	Programmeur	Datum

BLZ. 1–3	REGEL 4–6	C 7	A 8–11 / B 12–72	73–80
001	010		IDENTIFICATION DIVISION.	
	020		PROGRAM-ID. DATUMSHOW.	
	030	*	DIT ZOU EEN NOLLEDIG PROGRAMMA KUNNEN ZIJN.	
	040	*		
	050		ENVIRONMENT DIVISION.	
	060		CONFIGURATION SECTION.	
	070		SOURCE-COMPUTER. computer-naam.	
	080		OBJECT-COMPUTER. computer-naam.	
	090			
	100		DATA DIVISION.	
	110		WORKING-STORAGE SECTION.	
	120		77 DATUM-VAN-VANDAAG SIZE IS 6.	
	130			
	140		PROCEDURE DIVISION.	
	150		P1.	
	160		DISPLAY DATUM-VAN-VANDAAG.	
	170		STOP RUN.	
	180			
	190			
	200			

FIGUUR 12 COBOL coderingsformulier

apparaten dan noemen die gebruikt moeten worden voor het lezen en schrijven van bestanden?

5. Voert de compiler zelf de bewerkingen beschreven in de Procedure Division uit?

B2 CODERINGSREGELS

COBOL-programma's worden meestal geschreven in hoofdletters op coderingsformulieren (zie figuur 12). Alleen de eerste 72 kolommen (van dit formulier) mogen gebruikt worden. De terminaltypiste tikt de gegevens regel voor regel in, waarbij ze precies het coderingsformulier navolgt. De eerste zes kolommen mogen worden gebruikt om de regels achtereenvolgens te nummeren, maar wanneer men de gegevens op de terminal intikt, wordt die nummering gewoonlijk automatisch door de computer verzorgd. Gewoonlijk nummert men in eenheden van tien, waardoor later regels ingevoegd kunnen worden. De laatste acht kolommen van de regel (vanaf kolom 73) worden genegeerd door de compiler afgezien van het afdrukken ervan in de uitlijsting van het programma; bij gebruik van ponskaarten gebruikt men deze posities gewoonlijk voor een korte programmatitel.

Sommige woorden in een COBOL-programma hebben een bijzondere betekenis voor de compiler. Dit zijn de zogenaamde *sleutelwoorden* of *gereserveerde woorden*. 'DIVISION', 'DATA', 'SUBTRACT', 'TO' zijn gereserveerde woorden die we al zijn tegengekomen. Er bestaan meer dan 100 gereserveerde woorden: de programmeur mag ze alleen gebruiken binnen de precieze samenhang van de taalregels in COBOL.

De programmeur kan ook zelf woorden vormen voor de benaming van paragrafen, sections of gegevens. Voorbeelden van deze door de programmeur ingevoerde benamingen zijn SALARIS, BELASTING, BIJSLAG en DATUM-VAN-VANDAAG. Hoe op nog andere wijze van door de programmeur ingevoerde benamingen gebruik kan worden gemaakt zal later worden behandeld.

Er zijn twee *marges* op het coderingsformulier. Marge A begint bij kolom 8 en marge B bij kolom 12. COBOL-zinnen moeten altijd rechts van marge B geschreven worden (maar niet buiten kolom 72 natuurlijk). Alleen division-, section- en paragraafbenamingen schrijft men vanaf het begin van marge A. In de DATA DIVISION wordt deze marge A ook aangehouden voor een deel van de beschrijvingen van bestanden en records (zie hoofdstuk E). Paragraafnamen moeten gevolgd worden door een punt. Division- en sectionnamen moeten gevolgd worden door een spatie en respectievelijk de woorden DIVISION en SECTION en een punt. Er is voorgesteld om de twee marges in COBOL weg te doen, zodat zinnen, paragrafen enzovoort gewoon maar ergens kunnen worden

gecodeerd. Maar zelfs als uw compiler geen marges vereist, bevelen we toch aan ze te gebruiken waardoor u een overzichtelijke lay-out van uw programma krijgt. (Als u geen gebruik hoeft te maken van regelnummers, kunt u natuurlijk de marges laten beginnen bij de kolommen 1 en 5.)

Er moet minimaal één spatie tussen de woorden van een zin bestaan. Er is geen beperking naar boven. Een gehele regel kan desgewenst opengelaten worden: dit wordt vaak gedaan om de presentatie van de programma-uitlijsting te verbeteren. Het is een goede (programmeurs)gewoonte slechts één opdracht per regel te schrijven, hoewel de COBOL-regels deze beperking niet opleggen.

De compiler gaat uit van één spatie tussen kolom 72 op de ene regel en marge B op de volgende regel, d.w.z. dat u een woord eindigend in kolom 72 kunt schrijven en het volgende woord op de volgende regel te beginnen bij marge B. U kunt kolom 72 bij marge B voegen voor een afgebroken woord door een gedachtenstreep ('-') te plaatsen in de vervolgkolom (kolom 7) van de volgende regel. De gedachtenstreep staat aan het begin van de regel, wanneer een compiler wordt gebruikt die geen marges erkent. Er is voorgesteld dit voorschrift te veranderen, zodat de gedachtenstreep aan het einde van de regel komt te staan, net zoals in normale Engelse interpunctie; maar dit is nog geen standaard. Het gebruik van de vervolgkolom moet bij voorkeur worden vermeden. Het is verkieslijk het woord dat moet worden afgebroken weg te laten en het op de volgende regel in zijn geheel te noteren.

Opgaven

1. Hoe kan de compiler onderscheiden waar een division, section of paragraaf eindigt?
2. Hoe kan de compiler onderscheid maken tussen een paragraafnaam en het eerste (of enige) woord in een zin?
3. Hoeveel sectionnamen zijn er in het programma van figuur 12? Hoeveel paragraafnamen? Hoeveel zinnen in de Procedure Division?
4. Veronderstel dat u een COBOL-programma schrijft en u spelde ENVIRONMENT DIVISION als (a) ENVIRONMENTDIVISION of (b) ENVIRONMEN DIVISION. Zou de compiler (a) of (b) of beide of geen van beide vormen accepteren?
5. Studenten voor wie COBOL nieuw is krijgen soms een lijst van de gereserveerde woorden in handen en proberen dan een programma in het Engels te schrijven, waarbij ze die woorden gebruiken op een eigen wijze. Waarom is deze aanpak gedoemd te mislukken?

B3 VORMING EN INTERPUNCTIE VAN NAMEN

Door de programmeur in te voeren benamingen voor gegevens kunnen gevormd worden uit ieder alfabetisch teken van A t/m Z, de cijfers 0 t/m 9 en de gedachtenstreep ('-'), op voorwaarde dat er minimaal één alfabetisch teken in zit en de gedachtenstreep niet aan het begin of einde staat. Geoorloofde namen zijn dus ABC-2 en 1-XYZ. Maar het zijn geen *goede* namen, omdat ze niets betekenen. Een goede programmeur zal namen vormen die de gegevens beschrijven, bijvoorbeeld:

```
LOONLIJSTNUMMER
ARTIKELNUMMER
BELASTINGCODE
BIJSLAG.
```

Het gebruik van goede namen maakt het programma langer, maar dit is een kleine prijs voor overzichtelijkheid. Namen mogen niet meer dan 30 tekens bevatten.

Dezelfde regels zijn ook van toepassing op door de programmeur ingevoerde benamingen voor paragrafen. Alleen mogen paragraafnamen in hun geheel numeriek zijn. Het is echter een goede gewoonte om dit te vermijden, aangezien paragraafnamen eveneens zinvol behoren te zijn. Programmeurs schrijven soms P1., P2., P3., enzovoort, voor de opeenvolgende paragraafnamen. Dit systeem helpt zeker bij het lokaliseren van de paragraaf op de uitlijsting, maar een goede naam beschrijft ook de bewerkingen in de paragraaf, bijvoorbeeld:

```
P1-DATUM-ONTVANGST.
P2-AFDRUKKEN-FAKTUUR-KOP.
P3-AFDRUKKEN-FAKTUUR-REGEL.
```

De punt beëindigt een zin. Hoewel komma's en puntkomma's kunnen worden gebruikt om de leesbaarheid van COBOL te vergroten, bestaan er tamelijk strikte voorschriften dienaangaande. We raden de programmeur aan ze maar beter weg te laten en de leesbaarheid d.m.v. het gebruik van extra spaties of nieuwe regels te bevorderen. Punten (en eventueel andere interpunctietekens) behoren altijd te worden gevolgd door minimaal één spatie. (Er is voorgesteld om de strikte voorschriften m.b.t. interpunctie te versoepelen.)

Opgaven

1. Wat is fout aan de volgende zin?

```
ADD INTEREST,ONTVANGST TO KAPITAAL.
```

2. Welke benamingen van gegevens zijn ongeoorloofd?
   a) 7462 b) A c) P76 d) - e) -1 f) 2-2 g) A-1
   h) B-B-B-B-B i) 2-A j) FRED
   k) KONINGINELIZABETHDETWEEDE l) WORKING STORAGE

3. Schrijf op een coderingsformulier een paragraaf die het volgende zal uitvoeren: tel BIJSLAG op bij SALARIS en trek AOW en BELASTING ervan af. Schrijf dit eerst op als drie zinnen, vervolgens als één zin. Vorm een paragraafnaam en begin met regelnummer 002200.

4. Is deze zin geoorloofd?

BLZ. 1–3	REGEL 4–6	C 7	A 8–11	B 12–72
	010			SUBTRACT ORDERS
	020			FROM VOORRAAD.
	030			

FIGUUR 13 Opgave 4

B4 LITERALS EN FIGUURLIJKE CONSTANTEN

In de Data Division worden bestandsgegevens in de file-section en andere gegevens in de working storage-section beschreven. Ze worden dikwijls *variabelen* genoemd, omdat de inhoud van deze velden verandert wanneer nieuwe records worden ingelezen of wanneer er bewerkingen met die inhoud plaatsvinden.

Als een programma alle pagina's van een rapport die op de regeldrukker worden afgedrukt, zou nummeren, zou het paginanummer een variabele zijn. Aan het eind van ieder pagina zal het paginanummer met 1 verhoogd worden. De 1 is dus niet bedoeld als naam van een veld (de benaming zou trouwens niet correct zijn), waarvan de inhoud kan veranderen, maar geeft een *constante* weer. We spreken van een *literal* (letterlijk op te vatten symbool). 1 is een *numerieke literal*. Een numerieke literal kan gevormd worden door iedere reeks cijfers, met zo nodig een decimaalteken en voorafgegaan door een + of - teken. bijvoorbeeld:

```
1.0
+12
123.45
-7
-7.24612
```

Het decimaalteken mag niet het laatste teken zijn. Wanneer geen + of - teken wordt geplaatst, wordt een + aangenomen. Numerieke literals mogen niet meer dan 18 cijfers bevatten.

Er zijn ook *niet-numerieke literals*. Ze worden ingesloten door aanhalingstekens, bijvoorbeeld:

```
"FRED SMITH"
"BEGIN KARWEI"
"VOER DE VERKOOPGEGEVENS IN"
"VERKOOPRAPPORT"
" "
```

Let op dat een spatie telt als een teken. Een niet-numerieke literal mag maximaal 120 tekens lang zijn.

Het gebeurt vaak dat een niet-numerieke literal meer dan één regel van een coderingsformulier beslaat. In zo'n geval wordt de literal geschreven t/m kolom 72, een gedachtenstreepje wordt in kolom 7 van de volgende regel genoteerd, beginaanhalingstekens worden ergens rechts van marge B genoteerd en de rest van de literal wordt in de daaropvolgende kolom vervolgd. Slotaanhalingstekens worden genoteerd zoals eerder, bijvoorbeeld:

BLZ. 1–3	REGEL 4–6	C 7	A 8–11	B 12–72
	010			MOVE "VERKOOPOVERZICHT GROOTHANDEL VAN VORIG KWARTAAL, HALF-J
	020	-		"AAR EN JAAR" TO KOPREGEL.
	030			

FIGUUR 14 Het afbreken van een niet-numerieke literal

Als een niet-numerieke literal precies in kolom 72 eindigt, afgezien van de slotaanhalingstekens, dan wordt het vervolgd zoals hierboven is vermeld en de laatste aanhalingstekens komen dan in kolom 13 te staan. We zullen later echter nog laten zien dat we het voortzetten van een literal over meer dan een regel kunnen vermijden door een geschikte indeling van de working storage te maken.

Het onderscheid tussen numerieke en niet-numerieke gegevens kan belangrijk zijn in COBOL, aangezien de compiler deze beide gegevenssoorten verschillend kan behandelen.

Naar bepaalde constanten kan verwezen worden d.m.v. het gebruik van gereserveerde woorden. Deze staan bekend als *figuurlijke constanten*, die gebruikt kunnen worden in plaats van literals.

Figuurlijke constante	*De compiler vat dit op als één of meer*
ZERO, ZEROS, ZEROES	nullen
SPACE,SPACES	spaties
QUOTE, QUOTES	aanhalingstekens
HIGH-VALUE, HIGH-VALUES	de hoogst mogelijke waarde
LOW-VALUE, LOW-VALUES	de laagst mogelijke waarde (gewoonlijk nul)

Of de meervouds- dan wel de enkelvoudsvorm wordt gebruikt beïnvloedt niet de door de compiler aangenomen waarden. Wat hiervan de oorzaak is, zal duidelijk worden bij de behandeling van de MOVE opdracht.

Opgaven

1. Welke numerieke literals zijn ongeldig?
   a) +1.0   b) -7.7687796   c) 1,123,746   d) .7   e) $93.12
   f) 3.14157
2. Welke niet-numerieke literals zijn ongeldig?
   a) "FRD SMITH"   b) "127.6"   c) "ADD A TO B"
   b) "ZEROES"   e) "END OF JOB
3. Waarom mag het decimaalteken niet het laatste teken zijn in een numerieke literal?
4. Waarom hebben de ontwerpers van de taal de figuurlijke constante QUOTE ingevoerd?

ANTWOORDEN HOOFDSTUK B

Paragraaf B1

1. De punt laat zien waar de ene zin eindigt en de andere begint.
2. DATUM-VAN-VANDAAG is de naam van een veld in het geheugen. De gegevens die daar staan zouden worden afgedrukt (waarschijnlijk op de bedieningsschrijfmachine van de operator, maar dat is bijkomstig). Welke gegevens dat zijn, kunnen we niet weten.
3. a) 6 posities   b) Neen.
4. ENVIRONMENT DIVISION
5. Neen. De compiler maakt het doeltaalprogramma. Dit doeltaalprogramma laat de computer de instructies uit de Procedure Division uitvoeren. Als u dit niet begrijpt, herlees dan paragraaf A6.

Paragraaf B2

1. Division - door een nieuwe division
   Section - door een nieuwe section
   Paragraaf - door een nieuwe paragraaf
   } of doordat het einde van een programma wordt bereikt.
2. De paragraafnaam begint bij marge A; zinnen beginnen rechts van marge B. (Alternatief antwoord in geval de compiler niet met marges werkt: zinnen beginnen altijd met een COBOL-werkwoord.

3. Twee sections (Configuration, Working-Storage). Vier paragrafen (Program-Id, Source-Computer, Object-Computer, Para-1). Twee zinnen.
4. Geen van beide.
5. De gereserveerde woorden mogen alleen in de precieze samenhang toegestaan door, en binnen de beperkte betekenis geïmpliceerd door de regels van COBOL, gebruikt worden. Hun gebruik in het Engels kan daar geheel van verschillen.

Paragraaf B3

1. Geen spatie achter de komma. Beter is het de komma weg te laten.
2. a), d), e), f), l). In de laatste is namelijk een spatie opgenomen.
3. i)

```
002200 BEREKEN-NETTO-SALARIS.
002210     ADD BIJSLAG TO SALARIS.
002220     SUBTRACT AOW FROM SALARIS.
002230     SUBTRACT BELASTING FROM SALARIS.
```

   ii)

```
002210     ADD BIJSLAG TO SALARIS
002220     SUBTRACT AOW FROM SALARIS
002230     SUBTRACT BELASTING FROM SALARIS.
```

4. Ja, maar bizar.

Paragraaf B4

1. c), e).
2. e).
3. De compiler zou geen onderscheid kunnen maken tussen een punt die de zin beëindigt en een decimaalteken.
4. """" zou opgevat kunnen worden als: begin van een niet-numerieke literal, einde van een niet-numerieke literal, begin van een andere niet-numerieke literal. In feite staat COBOL deze constructie toe doordat twee aanliggende aanhalingstekens in een niet-numerieke literal opgevat worden als een enkelvoudig aanhalingsteken binnen die literal. Voorbeeld: "AB""CD" wordt opgevat als een beschrijving van de literal AB"CD.

C DE REKENKUNDIGE BEWERKINGEN

C1 DE PROCEDURE DIVISION - ADD EN SUBTRACT

Hoewel de Procedure Division als laatste in een COBOL-programma wordt gecodeerd, meen ik toch dat het beter is eerst enige kenmerken van deze division te behandelen alvorens de andere divisions te bekijken.

We zijn de opdrachten ADD en SUBTRACT al in voorbeelden tegengekomen. De opdracht ADD kan twee vormen aannemen:

```
1. ADD {identifier-1} [identifier-2] ... TO identifier-m [ROUNDED]
   ---  {literal-1   } [literal-2   ]     --               -------
       [ON SIZE ERROR imperative-statement]
           ---- -----

2. ADD {identifier-1} {identifier-2} [identifier-3] ...
   ---  {literal-1   } {literal-2   } [literal-3   ]
   GIVING identifier-m [ROUNDED] [ON SIZE ERROR imperative-statement]
   ------               -------      ---- -----
```

De onderstreepte woorden in hoofdletters in deze formele beschrijving zijn sleutelwoorden. Andere woorden in hoofdletters zijn facultatieve woorden die weggelaten kunnen worden zonder verandering van betekenis. Rechte haken geven aan dat het woord of de zin daarbinnen facultatief is, maar als ze opgenomen zijn veranderen ze in het algemeen de betekenis. Wanneer elementen boven elkaar gestapeld worden, moet een keuze worden gemaakt voor één van de elementen in de stapel. De *ellips* ... geeft aan dat het voorafgaande door haken of accolades omgeven element herhaald kan worden zo vaak als gewenst is. *Accolades* worden gebruikt om een verplicht element of stapel elementen in te sluiten. 'Identifier' is de naam van een veld; 'imperative statement' betekent een of meer imperatieve opdrachten.

Deze beschrijvingsmethode wordt algemeen gebruikt in handboeken over COBOL, dus het is de moeite waard om deze beschrijving te begrijpen. In dit boek veroorloof ik me echter enige vrijheid in de beschrijvingen om te vermijden dat de zaken

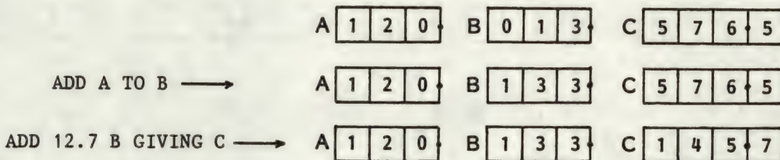
te ingewikkeld worden en in de verwachting dat het gezonde verstand van de student hem op het rechte pad zal brengen. De formele beschrijvingen worden volledig gegeven in Appendix D.

Geoorloofde ADD-opdrachten zijn bijvoorbeeld:

```
ADD A TO B.
ADD A TO B ROUNDED.
ADD 12.7 B GIVING C ON SIZE ERROR DISPLAY "ERR100".
ADD A B C GIVING D ROUNDED.
ADD 1272.437 TO B ROUNDED ON SIZE ERROR STOP RUN.
```

Merk op dat het woord TO niet opgenomen mag worden wanneer GIVING wordt gebruikt.

Het resultaat van de optelling staat in de operand uiterst rechts. De andere operands worden ongewijzigd gelaten. D.w.z.:



FIGUUR 15 Het resultaat van een ADD-bewerking

In figuur 15 is uitgegaan van de veronderstelling dat A en B in de Data Division zijn beschreven als velden met drie posities voor en geen positie na de decimale punt, terwijl C is beschreven met drie posities voor en één positie na de decimale punt.

Laten we nu eens veronderstellen dat C in figuur 15 in de Data Division als een veld zonder decimale plaatsen is beschreven. Na het uitvoeren van de laatste opdracht zou C dan slechts 145 bevatten; van het resultaat zou namelijk het decimale gedeelte (.7) worden afgekapt. *Afronding* van het resultaat zou een accurater antwoord opleveren: 146. Dit zou men hebben verkregen als de opdracht had geluid:

```
ADD 12.7 B GIVING C ROUNDED.
```

Het antwoord wordt dan afgerond naar het dichtstbijzijnde gehele getal voordat het in C wordt geplaatst. .5 zou naar boven afgerond worden. Dezelfde logica wordt toegepast wanneer de operand die het antwoord ontvangt *onvoldoende* decimale posities bevat om het gehele resultaat vast te houden.

Een '*size error*' komt voor wanneer het veld waarin het resultaat moet komen te staan onvoldoende ruimte heeft voor het geheeltallige gedeelte van het antwoord. Stel, dat we er in de opdracht

```
ADD 1272.437 TO B ROUNDED ON SIZE ERROR STOP RUN.
```

vanuit gaan dat B is geschreven met drie posities voor en achter de decimale punt en met een beginwaarde van -272.000. Na het uitvoeren van de instructie zou veld B +1000.437 moeten bevatten, maar omdat er slechts drie posities beschikbaar zijn voor de decimale punt, zal dat cijfer uiterst links afgekapt worden zodat B slechts 000.437 bevat. Wanneer ON SIZE ERROR is gespecificeerd, zal deze omstandigheid worden ontdekt en de imperatieve opdracht (STOP RUN in dit geval) uitgevoerd.

Opgaven

1.

A [0|9.0|0|0] B [0|1.0|0|5] C [2|3.0|0]

ADD A B GIVING C ——→ A [| . | |] B [| . | |] C [| . |]

FIGUUR 16 Opgave 1 - maak de tekening af

2. Bekijk de indeling van de SUBTRACT-opdracht in Appendix D. Welke van de volgende opdrachten zijn geoorloofd?
i) SUBTRACT A B ii) SUBTRACT A B C FROM D
iii) SUBTRACT B FROM ZERO GIVING FRED ROUNDED

3. De operand uiterst rechts in een ADD- of SUBTRACT-opdracht mag geen literal zijn. Waarom?

4. Schrijf op een coderingsformulier de opdrachten die het volgende doen: A := B+C-D+E-F+G. Alle operands zijn beschreven met twee posities voor en twee posities achter de decimale punt.

5. Is deze opdracht geoorloofd?

   ADD A AND B TO C.

6. Is deze opdracht geoorloofd?

   ADD A TO B GIVING C.

7. Als we bedenken dat A en B variabelen zijn die elke waarde kunnen aannemen in het bereik aangegeven door het aantal posities, waarom is de ADD-opdracht van figuur 16 dan onbevredigend?

C2 MULTIPLY EN DIVIDE

De eenvoudigste vorm van de vermenigvuldigingsopdracht is:

```
MULTIPLY {identifier-1} BY identifier-2
         {literal-1   }
```

Er bestaan keuzemogelijkheden voor GIVING, ROUNDED en SIZE ERROR, evenals bij ADD en SUBTRACT. Steeds wordt het resultaat van de vermenigvuldiging geplaatst in de operand uiterst rechts, terwijl de andere operands ongewijzigd blijven.

De opdracht tot deling is soortgelijk:

```
DIVIDE {identifier-1} INTO identifier-2
       {literal-1   }
```

Wederom bestaan er keuzemogelijkheden voor GIVING, ROUNDED en SIZE ERROR en staat het resultaat in de meest rechtse operand, terwijl de andere ongewijzigd blijven. Enkele oude compilers staan de gebruiker toe om te zeggen DIVIDE A BY B, waarbij de linker operand het resultaat bevat. Dit gebruik is echter geen standaard en is in strijd met de algemene regel m.b.t. de rekenkundige bewerkingen, die zegt dat het antwoord in de meest rechtse operand komt te staan. DIVIDE A BY B GIVING C gaat echter niet in tegen deze regel en is toegestaan in ANS COBOL.

Bij verreweg de meeste compilers kan men rechtstreeks de rest van een deling verkrijgen, bijvoorbeeld DIVIDE 3.1416 INTO CIRKEL GIVING DIAMETER ROUNDED REMAINDER REST-1 ON SIZE ERROR enzovoort. De rest van de deling komt nu in het veld REST-1 te staan (waarbij de rest wordt gedefinieerd als het verschil tussen deeltal en het produkt van de onafgeronde (mogelijk wel afgekapte) uitkomst en de deler).

Opgaven

1.

A	0	0	1	.	5	0	0

B	0	1	0	.	0	0

C	0	9	.	4	0

D	0	0	.	0	0

FIGUUR 16 Opgave 1

Welke inhoud hebben A, B, C en D na de volgende opdracht?

i) `MULTIPLY A BY B`
ii) `MULTIPLY A BY B ROUNDED`
iii) `DIVIDE B INTO A GIVING C REMAINDER D`
iv) `DIVIDE A INTO B GIVING C`
v) `DIVIDE A INTO B GIVING C REMAINDER D`

2. Wat is fout aan de volgende opdracht?

```
MULTIPLY STRAAL BY 3.1416.
```

3. Welk verschil in effect hebben deze twee opdrachten? Of is er geen verschil?

```
DIVIDE A INTO B GIVING C.
DIVIDE B BY A GIVING C.
```

4. Wat zou er gebeuren indien deze opdrachten worden uitgevoerd?

i)
```
SUBTRACT A FROM A
DIVIDE A INTO B GIVING C ON SIZE ERROR STOP RUN.
```

ii)
```
SUBTRACT A FROM A
DIVIDE A INTO B
ADD 1 TO A
DISPLAY A.
```

5. Uw compiler beschikt niet over de keuzemogelijkheid REMAINDER. U wilt toch de rest van de berekening GEMIDDELDE = SIGMA1/N weten. Een ander element SIGMA2 is beschreven met hetzelfde aantal posities als SIGMA1 (beide zijn gehele getallen). Schrijf opdrachten waardoor de rest van de deling in SIGMA2 geplaatst wordt. (We kunnen dit doen door voordeel te trekken van het feit dat afkapping zal plaatsvinden, wanneer de keuzemogelijkheid ROUNDED wordt weggelaten.)

C3 COMPUTE

De COMPUTE-opdracht is op enkele kleine compilers niet beschikbaar. Hij neemt deze vorm aan:

```
                                  { identifier-2          }
COMPUTE identifier-1 [ROUNDED] = { literal-1             }
                                  { arithmetic-expression }

[ON SIZE ERROR imperative-statement]
```

Deze opdracht heeft tot gevolg dat identifier-2 of literal-1 of het resultaat van de aritmetische expressie in identifier-1 wordt geplaatst. De aritmetische expressie is een combinatie van literals en/of identifiers met daartussen geplaatste rekenkundige bewerkingstekens. Deze bewerkingstekens zijn:

+ optellen
- aftrekken
/ delen
* vermenigvuldigen
** machtsverheffen

Bijvoorbeeld: een geoorloofde opdracht zou zijn:

```
COMPUTE ALPHA ROUNDED = BETA - BETA / N * N.
```

Voor mensen is deze opdracht dubbelzinnig (moeten we eerst aftrekken, of delen? d.w.z. (beta - beta)/n·n is niet hetzelfde als beta - (beta/n·n) noch is beta/(n·n) hetzelfde als (beta/n)·n). De compiler werkt echter volgens een strikt prioriteitenschema bij de uitwerking van een expressie:

Ten eerste **
Ten tweede * en /
Ten derde + en - .

En bij gelijke prioriteit (* en /, of + en -) vindt uitwerking plaats van links naar rechts.

Indien men van deze prioriteiten wil afwijken, kan men door haken aangeven dat het binnen de haken geplaatste gedeelte eerst uitgewerkt moet worden. Wanneer de uitdrukking ingewikkeld is, is het gebruik van haken aan te bevelen, ook al zijn ze niet strikt noodzakelijk, omdat men dan minder snel een fout maakt en de lezer de opdracht gemakkelijker begrijpt. Aan beide zijden van een rekenkundig bewerkingsteken (+, -, enzovoort) moet altijd een spatie worden gelaten.

Er is wellicht een voetangel voor de argeloze gebruiker van de compute-opdracht. Deze doet zich voor indien een size error in een deelberekening kan voorkomen. Wanneer bijvoorbeeld de reeks berekeningen een deling van een zeer groot getal door een zeer klein insluit, dan kan een size error optreden, hoewel het veld voor het resultaat van de totale berekening groot genoeg is om het uiteindelijke antwoord op te nemen. De SIZE ERROR zal dan echter toch in werking treden. Als de SIZE ERROR aan elke compute-opdracht wordt toegevoegd, loopt men dus niet het risico dat een size error optreedt en onontdekt blijft.

Opgaven

1. Is deze opdracht geoorloofd? COMPUTE A = B ROUNDED.
2. Welk bezwaar hebt u tegen deze opdracht?

```
COMPUTE GEMIDDELDE = ( A + B + C ) / N * 2.
```

3. De oppervlakte van een cirkel is $3.1416r^2$. Als de velden CIRKOPVL en STRAAL zijn gedeclareerd, bereken dan CIRKOPVL, eerst zonder Compute, daarna met.
4. Belastbaar inkomen bestaat uit JAARSAL en BONUS, minus pensioen en TOESLAGEN. Pensioen is niet gedeclareerd als een veld in de Data Division, maar het is altijd 5% van JAARSAL. BELASTING is 40% van belastbaar inkomen. Bereken BELASTING met één enkele compute-opdracht.
5. Veel compilers verzachten de regel m.b.t. spaties aan weerszijden van rekenkundige bewerkingstekens, maar eisen wel een spatie aan weerszijden van een minteken wanneer dat wordt gebruikt als rekenkundig bewerkingsteken. Waarom?

ANTWOORDEN HOOFDSTUK C

Paragraaf C1

1. C bevat 10.00; A en B ongewijzigd.
2. ii), iii). Merk op dat een figuurlijke constante een geoorloofd alternatief is voor een literal (zie paragraaf B4).
3. Omdat het veld dat het antwoord ontvangt rechts geplaatst moet worden.
4.

```
ADD B C E G GIVING A
ON SIZE ERROR
   DISPLAY "RESULTAAT TE GROOT"
   STOP RUN.
SUBTRACT D F FROM A
ON SIZE ERROR
   DISPLAY "RESULTAAT TE GROOT"
   STOP RUN.
```

5. Neen. (Enkele oude compilers staan deze constructie toe, maar ANS COBOL beperkt het gebruik van het woord AND tot een logische verbinding tussen twee voorwaarden. Dit wordt behandeld in hoofdstuk I.)
6. Neen.
7. De som van A en B kan maximaal 199.998 worden. Daarom zou men of C met drie posities voor de decimale punt moeten

beschrijven of SIZE ERROR moeten toevoegen. Mogelijk is ook de afwezigheid van het afrondingsmechanisme onbevredigend.

Paragraaf C2

1. i) B = 15.00; A, C en D ongewijzigd.
   ii) Zie i).
   iii) C = 00.15; A, B en D ongewijzigd.
   iv) C = 06.66; A, B en D ongewijzigd.
   v) C = 06.66; A en B ongewijzigd.
   D = 10.00 - (6.66x1.5) = 0.01.
2. Poogt antwoord in een literal op te slaan.
3. Geen verschil.
4. i) Computer zal stoppen.
   ii) Onbepaald, hangt af van computer. De meeste moderne computers stoppen gewoon met het programma zodra ze een nuldeling tegenkomen die niet van een SIZE ERROR is voorzien.
5.
```
DIVIDE N INTO SIGMA1 GIVING GEMIDDELDE ON SIZE ERROR etc. ...
MULTIPLY GEMIDDELDE BY N GIVING SIGMA2.
SUBTRACT SIGMA2 FROM SIGMA1 GIVING SIGMA2.
```
   Willen we GEMIDDELDE toch afgerond hebben, dan verkrijgen we dat door de volgende opdracht hieraan toe te voegen:
```
ADD 0.5 TO GEMIDDELDE
ON SIZE ERROR etc. ...
```

Paragraaf C3

1. Neen, alleen uitkomsten kunnen worden afgerond.
2. SIZE ERROR moet worden toegevoegd. Waarschijnlijk moet GEMIDDELDE afgerond worden.
3. i)
```
MULTIPLY STRAAL BY STRAAL GIVING CIRKOPVL ROUNDED
    ON SIZE ERROR imperative statement.
MULTIPLY 3.1416 BY CIRKOPVL ROUNDED
    ON SIZE ERROR imperative statement.
```
   ii)
```
COMPUTE CIRKOPVL ROUNDED = 3.1416 * STRAAL ** 2
    ON SIZE ERROR imperative statement.
```
4.
```
COMPUTE BELASTING ROUNDED =
        (JAARSAL + BONUS - TOESLAGEN - (JAARSAL * 0.05)) * 0.4
    ON SIZE ERROR imperative statement.
```
5. Als deze regel niet gemaakt was, dan kon de compiler geen onderscheid maken tussen één enkele identifier met een ingesloten koppelteken en twee identifiers, waarvan de een moet worden afgetrokken van de ander.

D OVERDRACHT VAN BESTURING

D1 GO TO; ENKELVOUDIGE VOORWAARDELIJKE OPDRACHT

De GO TO-opdracht wordt gevolgd door een paragraafnaam. De computer voert normaliter opdrachten uit in de volgorde waarin ze worden gecodeerd; wanneer hij een GO TO tegenkomt, vervolgt de uitvoering bij de eerste zin van de erbij gespecificeerde paragraaf.

Wanneer we bijvoorbeeld de besturing willen overbrengen naar een paragraaf die we LEES-VOLGENDE-KAART genoemd hebben, dan coderen we heel eenvoudig GO TO LEES-VOLGENDE-KAART.

De GO TO-opdracht wordt dikwijls afhankelijk gemaakt van een *voorwaarde*. Een voorwaarde wordt gespecificeerd door het doen beginnen van een zin met het woord IF. De eenvoudigste voorwaarde is gelijkstelling, bijvoorbeeld:

```
IF RECORD-TYPE = 1
   GO TO BEREKENING-1.
GO TO BEREKENING-OVERIGE.
```

Indien aan de voorwaarde voldaan wordt (d.w.z. RECORD-TYPE bevat 1), dan zal de besturing overgebracht worden naar de paragraaf die we BEREKENING-1 genoemd hebben. Indien niet aan de voorwaarde voldaan wordt, wordt aan de opdrachten in de voorwaardelijke zin stilzwijgend voorbijgegaan. De uitvoering vervolgt dan bij de volgende zin. In ons voorbeeld zal de volgende zin een onvoorwaardelijke opdracht naar BEREKENING-OVERIGE veroorzaken.

We kunnen iedere zin voorwaardelijk maken door aan het begin het woord IF en een voorwaarde te plaatsen. De tegenovergestelde voorwaarde wordt verkregen door het woord NOT, bijvoorbeeld:

```
IF RECORD-TYPE NOT = 1
   GO TO BEREKENING-OVERIGE.
GO TO BEREKENING-1.
```

Een bijzonder geval van de GO TO-opdracht ziet er als volgt uit:

<u>GO</u> <u>TO</u> PAR-A PAR B <u>DEPENDING</u> <u>ON</u> INVOERGEGEVEN

Indien het veld waarnaar verwezen wordt door de identifier INVOERGEGEVEN 1 bevat, dan zal de besturing overgaan naar de paragraaf die we PAR-A genoemd hebben; indien INVOERGEGEVEN 2 bevat, gaat de besturing over naar PAR-B. Ieder aantal (laten we zeggen, n) paragraafnamen kan zo gespecificeerd worden. Maar we moeten er wel zeker van zijn, dat het identificatieveld een geheel getal in de reeks 1 tot n bevat; zo niet, dan vindt geen overdracht van besturing plaats. Veel kleine compilers bieden niet de mogelijkheid van DEPENDING ON.

Opgaven

1. Een programmagedeelte ziet er als volgt uit:

```
    COMPUTE B = 0.
LUS.
    ADD A TO A.
    ADD 1 TO B.
    IF B = 2
       GO TO VERVOLG.
    GO TO LUS.
VERVOLG.
```

   De beginwaarde van A was 3. Welke waarde heeft A wanneer de besturing de paragraaf VERVOLG bereikt heeft?

2. Schrijf een opdracht waarbij A door B gedeeld wordt en de besturing bij nuldeling wordt overgebracht naar een paragraaf NULDELING.

3. De waarde van ELEMENT kan zijn: 12, 24 of 36. Als de waarde 12 is, wilt u de verwerking vervolgen bij P12; indien 24, bij P24 en indien 36, bij P36. Codeer deze logische gedachtengang a) met gebruikmaking van DEPENDING ON en b) zonder gebruikmaking van DEPENDING ON.

4. Schrijf opdrachten voor de vermenigvuldiging van een element VERMENIGVULDIGER met een element VERMENIGVULDIGTAL. Het gaat in beide gevallen om positieve, gehele getallen. Het antwoord moet komen te staan in een element RESULTAAT. Er is maar één probleem: u moet dit doen zonder gebruik te maken van MULTIPLY, DIVIDE of COMPUTE.

D2 ALTER

Deze opdracht wordt uitsluitend behandeld omdat u hem in programma's zoudt kunnen tegenkomen. Het gebruik ervan wordt sterk ontraden. Er is voorgesteld om deze opdracht uit een volgende

standaard van COBOL te verwijderen. De opdracht ALTER wordt gebruikt om de bestemming, zoals die in een GO TO-opdracht is gespecificeerd, te wijzigen. Wij hebben bijvoorbeeld een paragraaf:

```
FLIP-FLOP.
    GO TO FLIP.
```

In een later stadium van het programma kunnen we dan schrijven:

```
ALTER FLIP-FLOP TO PROCEED TO FLOP.
```

Na het uitvoeren van de ALTER-opdracht zal, bij het terugkeren van de besturing naar de paragraaf die we FLIP-FLOP genoemd hebben, de GO TO daarin uitgevoerd worden, alsof er stond GO TO FLOP.

Deze opdracht is enigszins overtollig, aangezien er altijd een ander middel bestaat om het gewenste resultaat te bereiken. Zoals gezegd, wordt het gebruik van deze opdracht sterk ontraden en wel omdat het buitengewoon moeilijk wordt de loop van het programma te volgen. (Om de waarheid te zeggen: een moderne theorie is dat men GO TO om dezelfde reden zou moeten vermijden; we komen hierop terug in de hoofdstukken K en L.) We kunnen soms een programma een beetje korter of sneller maken door ALTER te gebruiken, maar de verwarring die eruit voortvloeit weegt zwaar.

Opgaven

1. Waarom behoort u de ALTER-opdracht zoveel mogelijk te vermijden?
2. Prima, maar waarom maakt ALTER het moeilijk de loop van het programma te volgen?

D3 OPVATTINGEN OVER PERFORM

De PERFORM-opdracht is zondermeer de machtigste van COBOL. Een volledig begrip van zowel de opvattingen over als het praktisch gebruik van deze opdracht is wezenlijk voor de goede COBOL-programmeur. De eenvoudigste vorm van de opdracht is:

```
PERFORM paragraph-name-1 [THRU paragraph-name-2]
```

Laten we aannemen dat de opdracht

```
PERFORM BEREKENING-GEMIDDELDE
```

wordt uitgevoerd. De besturing wordt dan overgebracht naar de eerste zin van de paragraaf die we BEREKENING-GEMIDDELDE genoemd hebben, precies alsof een GO TO BEREKENING-GEMIDDELDE wordt uitgevoerd. Maar in tegenstelling tot GO TO *keert* de besturing, wanneer de laatste punt in BEREKENING-GEMIDDELDE is bereikt, *terug* naar de opdracht onmiddellijk na de PERFORM-opdracht.

Nog een voorbeeld:

```
PERFORM TOTAALCONTROLE.
ADD 1 TO CONTROLE-AANTAL.
```

Hier zal de paragraaf TOTAALCONTROLE eerst uitgevoerd worden, waarna het programma vervolgt met de zin ADD 1 TO CONTROLE-AANTAL.

Het kan voorkomen dat het ons niet uitkomt om de opdrachten die we door een perform willen laten uitvoeren in één paragraaf op te nemen. In zo'n geval gebruiken we de keuzemogelijkheid THRU door de eerste en laatste paragraaf van de reeks paragrafen te noemen die we willen laten uitvoeren. Bijvoorbeeld:

```
PERFORM TOTAALCONTROLE THRU EINDCONTROLE.
```

Hierin worden alle zinnen uitgevoerd vanaf de eerste zin in TOTAALCONTROLE tot en met de laatste zin in EINDCONTROLE. In ons voorbeeld wordt van PERFORM TOTAALCONTROLE THRU EINDCONTROLE wel gezegd dat het *in-line* gecodeerd is; de codering in de paragrafen TOTAALCONTROLE tot en met EINDCONTROLE wordt *out-of-line* genoemd. D.w.z.: de PERFORM-opdracht is er de oorzaak van dat de gespecificeerde out-of-line codering wordt uitgevoerd en dat de besturing terugkeert naar de volgende in-line opdracht.

Het meest waardevolle gebruik van de PERFORM-opdracht bestaat in het opsplitsen van een groot programma in kleinere onderdelen die gemakkelijk zijn te schrijven. Veronderstel dat we een programma willen schrijven dat de dag en de maand van het jaar inleest (worden door de operator ingevoerd), hieruit het aantal verstreken dagen berekent en het resultaat weergeeft. We gaan voor het gemak uit van maanden met 28 dagen. We kunnen de gehele in-line logica van het programma als volgt coderen:

```
PROCEDURE DIVISION.
P1-IN-LINE-PAR.
    PERFORM DATUM-ONTVANGST.
    PERFORM BEREKENING-DAGEN.
    PERFORM RESULTAAT-WEERGAVE.
    STOP RUN.
```

(Standaard COBOL vereist dat er een paragraafnaam staat voor de eerste zin in de Procedure Division, zelfs als er niet verder naar die paragraaf verwezen wordt, vandaar P1-IN-LINE-PAR.)

Nu we de *macro-logica* van ons programma ontworpen hebben, kunnen we onze aandacht richten op de codering van de *micro-logica* van iedere out-of-line paragraaf. We kunnen bijvoorbeeld vervolgen met:

```
DATUM-ONTVANGST.
    DISPLAY "WELKE MAAND? (VOER 2 CIJFERS IN)".
    ACCEPT MAAND.
    DISPLAY "DANK U, WELKE DAG? (2 CIJFERS)".
    ACCEPT DAG.
BEREKENING-DAGEN.
    SUBTRACT 1 FROM MAAND.
    MULTIPLY MAAND BY 28 GIVING TOT-DAGEN.
    ADD DAG TO TOT-DAGEN.
RESULTAAT-WEERGAVE.
    DISPLAY "DANK U, HET AANTAL DAGEN IS " TOT-DAGEN.
```

Natuurlijk is de volgorde waarin out-of-line paragrafen worden gecodeerd niet belangrijk voor de logica, hoewel dat de duidelijkheid bevordert indien zij gecodeerd worden in de rangorde waarin ze voor het eerst genoemd worden bij de in-line codering. (In dit voorbeeld is ervan uitgegaan dat de operator correct werkt en alleen cijfers intoetst. In de praktijk mag zoiets niet als vanzelfsprekend aangenomen worden; er moet dan gecontroleerd worden of er niet andere tekens ingevoerd worden en deze controles betekenen een uitbreiding van het programma.)

Opgave

1. Er bestaat een COBOL-regel, dat u binnen een reeks paragrafen die worden aangeroepen vanuit een ander programmadeel, niet een GO TO moet schrijven die een bestemming buiten die reeks heeft, als de daarop volgende besturing niet terugkeert naar de laatste paragraaf in die reeks. (Mocht u deze regel overtreden, dan zullen de meeste compilers niet aangeven dat u een fout gemaakt hebt.) Waarom zou een dergelijke praktijk ook ongewenst zijn vanuit het gezichtspunt van iemand die uw programma tracht te begrijpen?

D4 GENESTE PERFORMS

Veronderstel dat we het voorbeeldprogramma uit paragraaf D3 willen veranderen, zodat het met kalendermaanden werkt in plaats van met maanden van 28 dagen. Het hele programma zal duidelijk ingewikkelder worden, als het aantal dagen in overeenstemming

met dergelijke maanden moet worden berekend. We kunnen bij de oplossing echter nog steeds ons eerste ongecompliceerde logische patroon handhaven, door de opdrachten die we nodig hebben om het aantal dagen gelijkwaardig aan kalendermaanden te vinden in een aparte paragraaf te beschrijven. Deze paragraaf, die wij bijvoorbeeld DAGEN-VOORAFGAAND-AAN-MAAND kunnen noemen, kan dan bijvoorbeeld binnen de paragraaf BEREKENING-DAGEN worden aangeroepen:

```
 1 PROCEDURE DIVISION.
 2 P1-IN-LINE-PAR.
 3     PERFORM DATUM-ONTVANGST.
 4     PERFORM BEREKENING-DAGEN.
 5     PERFORM RESULTAAT-WEERGAVE.
 6     STOP RUN.
 7 DATUM-ONTVANGST.
 8     DISPLAY "WELKE MAAND? (2 CIJFERS)".
 9     ACCEPT MAAND.
10     DISPLAY "DANK U, WELKE DAG? (2 CIJFERS)".
11     ACCEPT DAG.
12 BEREKENING-DAGEN.
13     PERFORM DAGEN-VOORAFGAAND-AAN-MAAND.
14     ADD DAG TO TOT-DAGEN.
15 RESULTAAT-WEERGAVE.
16     DISPLAY "DANK U, HET AANTAL DAGEN IS " TOT-DAGEN.
17 DAGEN-VOORAFGAAND-AAN-MAAND.
18     IF MAAND = 1
19        COMPUTE TOT-DAGEN = 0.
20     IF MAAND = 2
21        COMPUTE TOT-DAGEN = 31.
       (etc., voor de overige maanden)
```

Laten we eens inventariseren wat we hebben gedaan. De opdracht in regel 4 veroorzaakt dat de besturing wordt overgedragen naar regel 12. En de opdracht in regel 13 naar regel 17. Wanneer de laatste paragraaf geheel is uitgevoerd, zal de besturing terugkeren naar regel 14; wanneer ook die is uitgevoerd keert de besturing terug naar regel 5.

Wanneer een out-of-line paragraaf zelf een perform-opdracht bevat, dan noemen we die perform-opdracht *genest*. Dit is het geval met de perform in regel 13. De paragrafen die door een geneste perform worden aangeroepen moeten òf geheel binnen òf geheel buiten de reeks paragrafen liggen waarnaar verwezen wordt door de in-line perform (in het voorbeeld zijn ze geheel buiten). Het nesten mag zo vaak als vereist plaatsvinden, nesten binnen nesten inbegrepen.

Opgaven

1. Doe alsof u de computer bent die het voorbeeldprogramma in deze paragraaf uitvoert. Lijst de paragraafnamen uit zoals u ze tekenkomt (dit wordt 'traceren' van het programma genoemd). Zorg voor inspringingen in de uitlijsting, wanneer een geneste PERFORM-instructie wordt uitgevoerd.

2. Schrijf op een coderingsformulier een gewijzigde versie van het programma in deze paragraaf, dat de dag, de kalendermaand en het jaar krijgt ingevoerd van de operator, het dagrangnummer berekent en het resultaat weergeeft. Het jaar wordt met vier cijfers ingevoerd en bij een schrikkeljaar heeft februari 29 dagen. Neem als gegeven aan dat het jaar een schrikkeljaar is als het een rest van 0 heeft bij deling door 4. (Idee: ga ervan uit dat februari 28 dagen heeft en tel 1 erbij als het een schrikkeljaar is en de maand na februari komt. Programmeer de logica voor het schrikkeljaar in een aparte out-of-line paragraaf.) Schenk tevens aandacht aan de nauwgezetheid van uw codering.

3. Wat is fout aan de volgende codering?

```
PROCEDURE DIVISION.
PARAGRAAF-1.
    PERFORM PAR-2 THRU PAR-5.
PAR-2.
    (etc.)
PAR-3.
    (etc.)
PAR-4.
    PERFORM PAR-5 THRU PAR-6.
PAR-5.
    (etc.)
PAR-6.
```

D5 EXIT

De EXIT-opdracht kan gebruikt worden in samenhang met PERFORM...THRU..., wanneer het wenselijk is halverwege terug te keren van de paragrafen die door de PERFORM-opdracht werden aangeroepen, d.w.z. dat dan één of meer paragrafen aan het eind van de subroutine niet meer worden uitgevoerd.

Veronderstel bijvoorbeeld dat een gedeelte van een programma er als volgt uitziet:

```
    PERFORM P1 THRU P4.
P1. {
P2. {
P3. {
P4. {
```

Laten we nu eens aannemen, dat zich in P2 een omstandigheid kan voordoen die voorschrijft dat P3 en P4 niet moeten worden uitgevoerd. Het probleem is: hoe vermijden we deze paragrafen zonder de regel m.b.t. GO TO te breken, die zei dat niet naar een bestemming buiten de aangeroepen reeks paragrafen gesprongen mag worden.

Voor dit geval bestaat de EXIT-opdracht. Deze wordt in een extra paragraaf aan het eind van de subroutine geplaatst, zodat een GO TO naar die paragraaf kan plaatsvinden, indien een vroege exit noodzakelijk is. EXIT is de enige opdracht in die paragraaf.

Ons probleem kan daarom als volgt worden opgelost:

```
    PERFORM P1 THRU P5
P1. {
P2. {
    IF omstandigheid GO TO P5.
P3. {
P4. {
P5.
    EXIT.
```

Voor deze paragraaf zijn geen opgaven.

ANTWOORDEN HOOFDSTUK D

Paragraaf D1

1. 12
2. DIVIDE B INTO A ROUNDED ON SIZE ERROR GO TO NUL-DELING.

3. a)
```
DIVIDE 12 INTO ELEMENT.
GO TO P12 P24 P36 DEPENDING ON ELEMENT.
```

b)
```
IF ELEMENT = 12
   GO TO P12.
IF ELEMENT = 24
   GO TO P24.
GO TO P36.
```

(Merk op dat we ervan uitgaan dat de waarde van ELEMENT geen andere kan zijn dan 12, 24 of 36; bij enige twijfel zou de laatste opdracht gecodeerd zijn: IF ELEMENT = 36 GO TO P36. en gevolgd worden door opdrachten om de fout te behandelen.)

4. (U zou kunnen starten met de opdracht: COMPUTE RESULTAAT = ZERO als er enige twijfel bestond of de beginwaarde van RESULTAAT nul was.)

```
TEL-OP.
    IF VERMENIGVULDIGER NOT = ZERO
       SUBTRACT 1 FROM VERMENIGVULDIGER.
       ADD VERMENIGVULDIGTAL TO RESULTAAT.
       GO TO TEL-OP.
```

Houdt uw antwoord ook rekening met het geval waarbij één van de operands nul is?

Paragraaf D2

1. Omdat het moeilijkheden oplevert bij het volgen van de loop van het programma.
2. Wat u ziet geschreven op het coderingsformulier of de uitlijsting is niet noodzakelijk dat wat de computer zal doen.

Paragraaf D3

1. Wanneer u een perform-opdracht tegenkomt, verwacht u dat de besturing terugkeert naar de volgende in-line opdracht. Dit is niet het geval, indien een GO TO is geschreven met een bestemming buiten de reeks van de door de perform-opdracht aangeroepen paragrafen, tenzij een andere GO TO wordt ingevoegd die de besturing weer terugvoert in de reeks. Het is echter aan te bevelen nooit een GO TO te gebruiken die de besturing buiten de reeks van de door een perform-opdracht aangeroepen paragrafen brengt.

Paragraaf D4

1.
```
P1
    DATUM-ONTVANGST
    BEREKENING-DAGEN
       DAGEN-VOORAFGAAND-AAN-MAAND
    RESULTAAT-WEERGAVE
```

2.
```
PROCEDURE DIVISION.
P1-IN-LINE-PAR.
    PERFORM DATUM-ONTVANGST.
    PERFORM BEREKENING-DAGEN.
    PERFORM RESULTAAT-WEERGAVE.
    STOP RUN.
DATUM-ONTVANGST.
    DISPLAY "WELKE DAG (VOER 2 CIJFERS IN)".
    ACCEPT DAG.
    DISPLAY "WELKE MAAND (2 CIJFERS)".
    ACCEPT MAAND.
 DISPLAY "WELK JAAR? (4 CIJFERS)".
    ACCEPT JAAR.
BEREKENING-DAGEN.
    PERFORM BEPALING-MAANDDAGEN.
    ADD DAG TO TOT-DAGEN.
RESULTAAT-WEERGAVE.
    DISPLAY "HET DAGRANGNUMMER " TOT-DAGEN.
BEPALING-MAANDDAGEN.
    IF MAAND = 1
       COMPUTE TOT-DAGEN = 0.
    IF MAAND = 2
       COMPUTE TOT-DAGEN = 31.
    IF MAAND = 3
       COMPUTE TOT-DAGEN = 59
       PERFORM SCHRIKKEL-CONTROLE.
    IF MAAND = 4
       COMPUTE TOT-DAGEN = 90
       PERFORM SCHRIKKEL-CONTROLE.
    etc.
SCHRIKKEL-CONTROLE.
    DIVIDE 4 INTO JAAR REMAINDER HULPVELD
    IF HULPVELD = ZERO
       ADD 1 TO TOT-DAGEN.
```

(Grijp deze gelegenheid aan om uw nauwgezetheid in het coderen te toetsen:

- Begint u de paragraafnamen in kolom 8 (als uw compiler dat tenminste vereist)?
- Codeert u slechts 1 teken per hokje en gebruikt u hoofdletters?
- Begint u zinnen in of rechts van kolom 12?

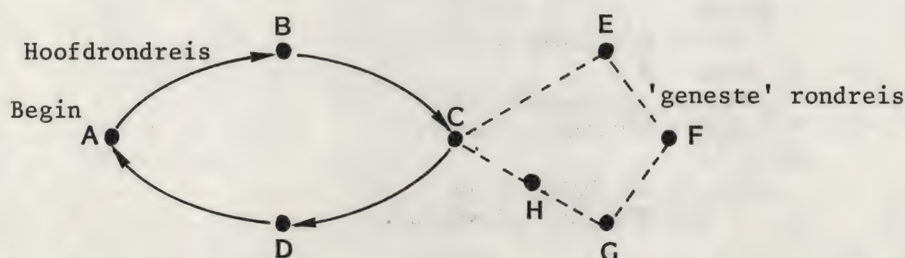
- Zet u een punt achter iedere paragraafnaam?
- Zet u een punt aan het einde van iedere paragraaf en aan het einde van iedere IF-opdracht?
- Heeft u rekening gehouden met plaatselijke gewoontes (afspraken) voor de letter O en de nul?
- Plaatst u een spatie aan weerszijden van het = teken, en na punten en komma's?)

3. De geneste perform in PAR-4 wil een reeks paragrafen uitvoeren die noch geheel intern noch geheel extern liggen t.o.v. de reeks paragrafen die door de PERFORM-opdracht in PARAGRAAF-1 worden aangeroepen.

Veel studenten vinden het achterliggende idee van de PERFORM, vooral de geneste PERFORM, moeilijk te begrijpen. Maak u niet bezorgd, als dit ook op u van toepassing is; er is tijd genoeg om u dit idee eigen te maken. Mogelijkheden voor meer oefening met PERFORM worden later geboden; intussen kan deze analogie wellicht het begrip bevorderen.

Een PERFORM-opdracht lijkt op een rondreis per bus, in die zin dat u altijd eindigt waar u begonnen bent. De paragrafen die worden uitgevoerd zijn als de plaatsen waar u stopt tijdens de rondrit.

Veronderstel dat u bij een van deze pleisterplaatsen uw bus verlaat en een andere rondreis maakt. Waarna u zich weer in uw eerste bus voegt. U bent dan op een 'geneste' rondreis geweest, analoog aan een geneste perform.

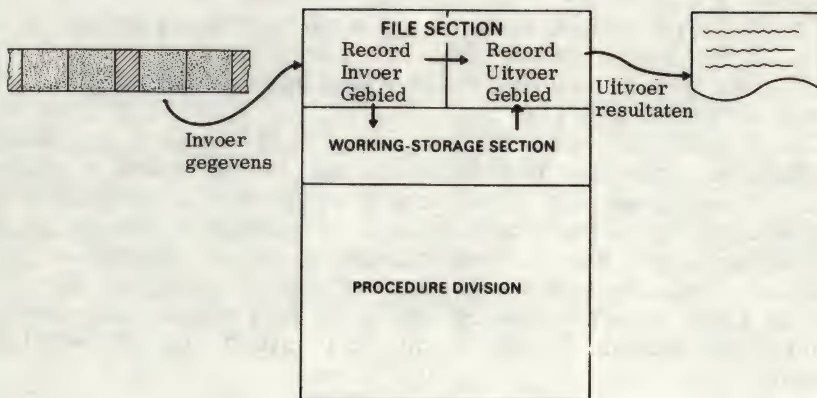


FIGUUR 18 Geneste ronreis per bus

We kunnen de analogie nog uitbreiden om de regels voor het nesten erin te betrekken. De plaatsen die u bezoekt bij uw tweede rondreis moeten verschillen van die bij uw bezoek van de eerste rondreis (zie figuur 18) of ze moeten een deelverzameling vormen van de plaatsen die u bezocht bij de eerste rondreis. Als u tijdens uw tweede rondreis nieuwe plaatsen aandoet mag u in die reis geen plaatsen van de eerste reis aandoen.

E DE DATA DIVISION

E1 INLEIDING



FIGUUR 19 De inhoud van het geheugen tijdens de uitvoering van een programma

Figuur 19 illustreert de basisprincipes die ten grondslag liggen aan een COBOL-programma. Een record uit een invoerbestand kan ingelezen worden in een geheugengebied onder besturing van de programmaprocedures (voor dit doel bestaat een opdracht READ). Een record voor een uitvoerbestand kan alleen weggeschreven worden vanuit een uitvoergebied voor records (WRITE bestaat voor dit doel). Het is noodzakelijk, zelfs wanneer we hetzelfde record uitschrijven als werd ingevoerd, dat record intern in het geheugen te verplaatsen, d.w.z. van het invoergebied voor records naar het uitvoergebied voor records (hiervoor bestaat MOVE; de pijlen in figuur 19 laten zien dat de verplaatsing direct van het invoer- naar het uitvoergebied voor records kan plaatsvinden, of via working storage (het kladgeheugen)).

De Data Division kan worden verdeeld in vijf sections, die ieder een specifiek doel hebben. Maar eenvoudige COBOL-programma's hebben slechts twee sections nodig, namelijk de FILE SECTION en de WORKING-STORAGE SECTION.

In de FILE SECTION beschrijven we de bestanden die door het programma worden gebruikt. De compiler benut deze informatie om geheugengebieden te reserveren voor de invoer- en uitvoer-records. Veronderstel bijvoorbeeld dat we een stapel kaarten hebben met een getal in de eerste twee kolommen van iedere kaart en we willen de som van alle getallen kennen. We beschrijven de kaarten als een bestand. Iedere kaart is dan een record in het bestand en de eerste twee kolommen zijn het veld of gegevens-element waarin we zijn geïnteresseerd. Wanneer we dit kaarten-bestand in de FILE SECTION beschrijven, zal de compiler een invoergebied voor de records reserveren om de gegevens van de kaart, wanneer die wordt ingelezen, op te slaan.

De som die we willen bijhouden als de kaarten worden ingelezen, is wel een gegevenselement maar geen deel van het bestand. Zulke gegevenselementen worden beschreven in de WORKING-STORAGE SECTION.

In de Data Division beschrijven we dus de records en velden of gegevenselementen die in het programma moeten worden gebruikt en geven we een naam aan ieder record en veld. De compiler wendt deze informatie aan om de plaats van de gegevens in het geheugen in kaart te brengen. Wanneer we vervolgens in de Procedure Division naar de namen van gegevens verwijzen, dan kan de compiler de twee namen verbinden en onze procedure-opdrachten vertalen in machinecode die op de juiste plaatsen zijn uitwerking zal hebben.

Opgaven

1. Welke gegevensnamen moeten in de Data Division van het programma, waarvan u een deel van de Procedure Division in figuur 10 aantreft, zijn beschreven?

2. De compiler geeft u een foutmelding die we een fatale diagnostiek noemen, als hij uw programma niet in machinecode kan vertalen door een fout van uw kant. Indien u in de Procedure Division zou verwijzen naar een gegevenselement, maar zou verzuimen het in de Data Division te beschrijven, zou de compiler uw programma dan een fatale diagnostiek geven, denkt u?

3. Als u een gegevenselement in de Data Division benoemt, maar helemaal niet ernaar verwijst in de Procedure Division, zou de compiler dan een fatale diagnostiek geven?

4. Een programma moet een magneetschijfbestand inlezen en o.a. tellen hoeveel records erop staan.

a) In welke section zou u de records van de magneetschijf beschrijven?
b) Welk ander veld zou u nodig hebben en waar zou u het beschrijven?

E2 DE FILE SECTION

We vertellen de compiler dat we de File Section beginnen door de naam van de section onmiddellijk na de naam van de division te coderen, dus:

```
DATA DIVISION.
FILE SECTION.
```

Ieder bestand dat door het programma wordt gebruikt wordt verder in twee niveaus beschreven; een *bestandsbeschrijving* of *File Definition* (FD niveau) en één of meer *recordbeschrijvingen* of *Record Descriptions* (01 niveau).

Bestandsbeschrijvingen kunnen zeer ingewikkeld zijn: voor een complete beschrijving zouden we het technische handboek van de computerfabrikant moeten raadplegen. De belangrijke kenmerken die we nu moeten kennen worden gegeven in deze verkorte vorm:

```
FD filename [BLOCK CONTAINS integer RECORDS]
            LABEL RECORDS ARE {STANDARD}
                              {OMITTED }
            [VALUE OF IDENTIFICATION IS literal]
```

De letters FD, die staan voor File Definition, worden bij compilers die met de marges werken gecodeerd in marge A. De rest van de beschrijving wordt rechts van marge B genoteerd. De bestandsnaam verzinnen we zelf, in overeenstemming met de regels voor het vormen van namen (paragraaf B3). De facultatieve passage BLOCK CONTAINS ... kan worden gebruikt om het aantal records per blok (paragraaf A4) in magneetband- en schijfbestanden te specificeren.

Indien LABEL RECORDS ARE STANDARD is gespecificeerd, dan moet het bestand over het algemeen (raadpleeg uw handleiding) een kop- en staartlabel hebben (paragraaf A4). Wanneer labels standaard zijn, dan mag een VALUE OF IDENTIFICATION (IDENTIFICATION is geen standaardwoord. Soms gebruikt men FILE-ID of alleen maar ID) worden gespecificeerd. Als de passage inderdaad is gespecificeerd, dan wordt met de literal bedoeld de naam die is vastgelegd op de koplabel als identificatiesymbool van het bestand.

Indien er meer dan één recordsoort in een bestand is opgenomen, krijgt iedere recordsoort zijn eigen gedetailleerde recordbeschrijving onder de bestandsbeschrijving waarin de recordsoort thuishoort.

Voorbeeld: Een programma leest records uit twee bestanden; het ene is een kaartenbestand, het andere een magneetbandbestand. Het kaartenbestand heeft geen labelrecords en bevat twee verschillende recordsoorten; de ene soort zullen we DEBETPOSTEN noemen, de andere CREDITPOSTEN. Het magneetbandbestand heeft standaardlabels met de identificatie 'STAMBESTAND' die is vastgelegd op de koplabel. De blokkingsfactor (het aantal records in een blok) van de records op de magneetband is 3. Dit bestand heeft slechts één recordsoort.

```
DATA DIVISION.
FILE SECTION.
FD  KAARTBESTAND LABEL RECORDS OMITTED
(hier volgt de recordbeschrijving van DEBETPOSTEN)
(hier volgt de recordbeschrijving van CREDITPOSTEN)
FD  BANDBESTAND BLOCK CONTAINS 3 RECORDS
                LABEL RECORDS STANDARD
                VALUE OF IDENTIFICATION "STAMBESTAND".
(hier volgt de recordbeschrijving van BAND-REC)
```

De LABEL-passage is nu dwingend voorgeschreven in standaard COBOL, maar er is voorgesteld om deze facultatief te maken. Op de meeste moderne computers kan de identificatie van het bestand (evenals van andere kenmerken, zoals blokomvang) worden bewerkstelligd ten tijde van de uitvoering van het programma d.m.v. commando's naar het bedieningssysteem (deze commando's maken geen deel uit van COBOL en zijn niet gestandaardiseerd). Wanneer deze procedure wordt gevolgd, behoeft de bestandsbeschrijving alleen FD bestandsnaam te bevatten.

Opgaven

1. Een magneetschijfbestand met 'ARTIKELBEST' vastgelegd op een standaard koplabel, heeft records (die we ARTREC noemen) in blokken van 100. Noteer de bestandsbeschrijving waarbij u het bestand INVOERBESTAND noemt.
2. Een bestand genaamd AFDRUKBESTAND moet worden uitgevoerd op de regeldrukker. De records moeten AFDRUK-REGEL worden genoemd. Noteer de bestandsbeschrijving (geen labels).

E3 DE RECORDBESCHRIJVING

Een recordbeschrijving is in wezen een reeks beschrijvingen met de vorm

```
level-number data-name [PICTURE IS picture-string] .
```

Bij het rangnummer 01, dat altijd wordt gecodeerd in marge A, geven we de naam van het gehele record aan. De rangnummers 02 t/m 49, gecodeerd vanaf marge B, worden gebruikt om de velden die het record omvat te beschrijven. Dus rangnummer 02 duidt een veld binnen een record aan. Een veld met rangnummer 02 kan worden opgesplitst in subvelden die ieder een rangnummer 03 hebben, enzovoort. (De nul aan het begin in een rangnummer met één cijfer is facultatief, maar de meeste programmeurs gebruiken hem wel.)

De PICTURE-passage wordt gebruikt om het aantal en de soort tekens in het veld te beschrijven. Zo betekent PICTURE XXX drie tekens van welke soort ook (alfanumeriek); PICTURE 9999 betekent vier tekens voor numerieke gegevens; PICTURE A betekent één teken voor alfabetische gegevens (PICTURE A wordt zelden gebruikt omdat PICTURE X ook alfabetische tekens insluit); PICTURE XX99 twee alfanumerieke tekens gevolgd door twee numerieke tekens, enzovoort. Om het schrijven van lange reeksen tekens voor grote elementen te vermijden kunnen we het aantal keren dat een teken herhaald zou moeten worden ook tussen haakjes plaatsen. Zo kunnen we bijvoorbeeld in plaats van PICTURE XXXXXXXXXX schrijven PICTURE X(10); PICTURE 9(6) i.p.v. PICTURE 999999; PICTURE X(5)9(4) i.p.v. PICTURE XXXXX9999. Het woord PICTURE mag worden afgekort tot PIC.

Als een stapel kaarten een getal bevat in de eerste twee kolommen en de rest oningevuld blijft, dan kan het kaartrecord als volgt worden beschreven:

```
01  KAARTGEGEVENS.
    02 GETAL      PICTURE 99.
    02 ONGEBRUIKT PICTURE X(78).
```

De compiler zou hieruit afleiden dat KAARTGEGEVENS de naam van een record is; dat GETAL de naam is van een veld dat uit twee numerieke tekens bestaat (de eerste twee van het record) en dat ONGEBRUIKT de naam voor de volgende 78 alfanumerieke tekens van het record is. Voor zo'n veld dat we niet verder gebruiken en waarnaar we dus in de Procedure Division niet verder verwijzen, hoeven we zelf echter geen naam te bedenken; we kunnen er het gereserveerde woord FILLER voor gebruiken.

We kunnen nu een compleet voorbeeld van zowel bestands- als recordbeschrijving geven:

```
DATA DIVISION.
FILE SECTION.
FD  KAARTBESTAND LABEL RECORDS OMITTED.
01  KAARTGEGEVENS.
    02 GETAL  PICTURE 99.
    02 FILLER PICTURE X(78).
```

Merk op dat de KAARTGEGEVENS zelf geen picture krijgt. De compiler zal de lengte van KAARTGEGEVENS berekenen aan de hand van de velden die KAARTGEGEVENS bevat; er wordt aangenomen dat het soort gegevens van een groepsveld alfanumeriek is (hoe de elementaire velden van het groepsveld dan ook beschreven zijn). Voor ieder groepsveld geldt dus: het is even lang als de som van de velden die het omvat, is alfanumeriek en krijgt geen picture.

Opgaven

1. Een bestand (naam: OVERZICHT-BESTAND) moet uitgeschreven worden naar de regeldrukker. De drukregels hebben vijf velden, namelijk (1) een numeriek rekeningnummer van 6 cijfers; (2) een ongebruikt veld van 3 tekens; (3) een alfanumerieke naam van 9 tekens; (4) een tweede ongebruikt veld van 3 tekens en (5) een adres van 26 tekens. Geef de volledige Data Division beschrijving van dit bestand. Welke lengte zal de compiler aan het record toekennen?
2. Geef een volledige recordbeschrijving voor dit recordontwerp:

Record-naam	DEBET-RECORD							
rangnr. 02	REKENINGNUMMER			DATUM			FACTUURNR.	BEDRAG
rangnr. 03	GEBIEDSCODE	SERIENR.	CONTROLE CIJFER	DAG	MAAND	JAAR		
soort gegevens	X	9999999	X	99	99	99	99999999	99999999

FIGUUR 20 Opgave 2

3. Maak een diagram (zoals in figuur 20) van deze record-beschrijving:

```
01  ARTIKEL-RECORD.
    02 ARTIKELNR.
       03 EIGENSCHAPS-CODE.
          04 MATERIAAL PICTURE X.
          04 VORM      PICTURE X.
       03 DIMENSIES.
          04 LENGTE    PICTURE 99.
          04 GEWICHT   PICTURE 9.
       03 VOLGNR       PICTURE 9999.
    02 OMSCHRIJVING    PICTURE X(10).
```

ANTWOORDEN HOOFDSTUK E

Paragraaf E1

1. SOM, TELLER, GETAL.
2. Ja.
3. Neen.
4. a) In de File Section.
   b) Een veld voor het optellen, in de Working-Storage Section.

Paragraaf E2

1.
```
FD  INVOERBESTAND BLOCK CONTAINS 100 RECORDS
                  LABEL RECORDS STANDARD
                  VALUE OF IDENTIFICATION "ARTIKELBEST".
(hier volgt de recordbeschrijving van ARTREC)
```

2.
```
FD  AFDRUKBESTAND LABEL RECORDS OMITTED.
(hier volgt de recordbeschrijving van AFDRUK-REGEL)
```

Heeft u OMITTED correct gespeld? En heeft u een punt aan het einde van de bestandsbeschrijving geplaatst en nergens anders?

Paragraaf E3

1.
```
DATA DIVISION.
FILE SECTION.
FD  OVERZICHT-BESTAND LABEL RECORDS OMITTED.
01  REGEL.
    02 REKENINGNUMMER PICTURE 9(6).
    02 FILLER         PICTURE XXX.
    02 NAAM           PICTURE X(9).
    02 FILLER         PICTURE XXX.
    02 ADRES          PICTURE X(26).
```

De recordlengte is 47 tekens.

2.
```
01  DEBET-RECORD.
    02 REKENINGNUMMER.
       03 GEBIEDSCODE     PICTURE X.
       03 SERIENR         PICTURE 9(7).
       03 CONTROLE-CIJFER PICTURE X.
    02 DATUM.
       03 DAG             PICTURE 99.
       03 MAAND           PICTURE 99.
       03 JAAR            PICTURE 99.
    02 FAKTUURNR          PICTURE 9(8).
    02 BEDRAG             PICTURE 9(8).
```

Het is niet strikt nodig hogere rangnummers (03 enzovoort) te laten inspringen, zoals ik hier heb gedaan. Maar het wordt gewoonlijk gedaan om de overzichtelijkheid te bevorderen.

3.

<table>
<tr><td>Record-naam</td><td colspan="6">ARTIKEL-RECORD</td></tr>
<tr><td>rangnr. 02</td><td colspan="5">ARTIKELNR</td><td rowspan="3">OMSCHRIJVING</td></tr>
<tr><td>rangnr. 03</td><td colspan="2">EIGENSCHAPS-CODE</td><td colspan="2">DIMENSIES</td><td rowspan="2">VOLGNR.</td></tr>
<tr><td>rangnr. 04</td><td>MATERIAAL</td><td>VORM</td><td>LENGTE</td><td>GEWICHT</td></tr>
<tr><td>soort gegevens</td><td>X</td><td>X</td><td>99</td><td>9</td><td>9999</td><td>X(10)</td></tr>
</table>

FIGUUR 21 Antwoord bij opgave 3

F VERVOLG DATA DIVISION

F1 DE WORKING-STORAGE SECTION

Gegevenselementen die geen deel uitmaken van een bestand worden beschreven in de Working-Storage Section, waarbij we gebruik kunnen maken van een speciaal rangnummer 77. Indien we bijvoorbeeld een totaal bestaande uit 3 cijfers willen bijhouden, kunnen we schrijven:

```
WORKING-STORAGE SECTION.
77  TOTAAL PICTURE 999.
```

Vanuit het standpunt van de logica van het programma kan een element met rangnummer 77 altijd worden vervangen door een element met rangnummer 01. Het hierboven genoemde voorbeeld kan dus worden geschreven:

```
01  TOTAAL PICTURE 999.
```

Het gebruik van een 01-element suggereert wel dat het zal worden opgesplitst in verdere ondergeschikte elementen, of dat het zal worden ingevoerd of uitgevoerd als een record op een randapparaat, maar dit is niet noodzakelijk. Het element met rangnummer 77 kan niet worden onderverdeeld noch gebruikt als een record. Sommige computerleveranciers hebben gebruik gemaakt van dit verschillende gebruik door ervoor te zorgen dat elementen met rangnummer 77 in het geheugen worden opgeslagen op een economisch meer verantwoorde of efficiëntere wijze dan elementen met rangnummer 01.

Zeer ervaren programmeurs proberen ordening in de workingstorage te scheppen door alle verwante elementen tesamen te groeperen als elementen met rangnummer 02, die dan ondergeschikt zijn aan een record naar hun keuze met rangnummer 01. Als deze gewoonte wordt opgevolgd, zal het programma geen elementen met rangnummer 77 bevatten. Er is voorgesteld om rangnummer 77 in standaard COBOL af te schaffen.

We mogen nooit zonder meer veronderstellingen maken over de beginwaarde van een element in de working storage. Als we willen dat het element bij de aanvang nul bevat, dan kunnen we dit specificeren met een VALUE-passage,

```
02  TOTAAL PICTURE 999 VALUE ZERO.      of
02  TOTAAL PICTURE 999 VALUE 0.
```

We kunnen een element op iedere beginwaarde zetten (mits natuurlijk het element groot genoeg is voor die waarde). Wanneer we bijvoorbeeld van plan zijn de pagina's van een rapport dat door de regeldrukker wordt afgedrukt te pagineren, dan zouden we een paginateller op een beginwaarde 1 kunnen zetten:

```
01  PAGINA-TELLER PICTURE 999 VALUE 1.
```

Een niet-numerieke literal kan worden gebruikt om een niet-numeriek element een beginwaarde te geven, bijvoorbeeld:

```
01  PAGINA-KOP PICTURE X(46) VALUE "VERKOOP RAPPORT OVER
-     " DE LAATSTE TWAALF MAANDEN".
```

Als de niet-numerieke literal in een VALUE-passage korter is dan de lengte van het element wordt deze literal links aangesloten in het element en worden de overblijvende posities aan de rechterkant met spaties opgevuld.

De VALUE-passage mag niet worden gebruikt in de File Section. Elementen met rangnummer 77 en 01 mogen in iedere volgorde in working storage worden gecodeerd. Het is dikwijls zinvol een record op te bouwen (d.w.z. met gegevens te vullen) in de Working-Storage Section en vervolgens het te verplaatsen naar een record in de File Section om uit te schrijven. Dit is vooral zinvol, indien een veld in het record constanten bevat, aangezien we een element geen beginwaarde in de File Section mogen geven. Veronderstel dat we voor de volgende uitvoerregel een record willen beschrijven:

(vert.-nr.) : (naam) : (verkoop deze maand) : (verk. tot datum)

De dubbele punten en spaties zijn constant, de andere elementen zijn variabel. Het record kan in kaart worden gebracht in working storage op de volgende wijze:

```
01  REGEL.
    02 VERTEGENW-NR        PICTURE 9(4).
    02 FILLER              PICTURE X(3) VALUE " : ".
    02 NAAM                PICTURE X(20).
    02 FILLER              PICTURE X(3) VALUE " : ".
    02 VERKOOP-DEZE-MAAND  PICTURE 9(5).
    02 FILLER              PICTURE X(3) VALUE " : ".
    02 VERKOOP-TOT-DATUM   PICTURE 9(6).
```

Het gereserveerde woord FILLER wordt hier gebruikt om de constanten te benoemen. Het is er alleen om de programmeur de moeite te besparen van het bedenken van nieuwe namen waarnaar niet afzonderlijk wordt verwezen in de Procedure Division. Het woord FILLER kan net zo vaak als nodig is worden gebruikt, maar uitsluitend voor elementaire elementen. Aangezien FILLER alleen aangeeft dat er niet naar dit element zal worden verwezen heeft CODASYL voorgesteld dit woord facultatief te maken: als een element geen naam heeft, wordt het dan opgevat als een FILLER.

Opgaven

1. Wat is fout aan deze codering?

```
FILE SECTION.
FD  KAARTBESTAND LABEL RECORDS OMITTED.
01  KAART PICTURE X(80) VALUE SPACES.
```

2. Beschrijf een Working-Storage Section die u in staat stelt de kop 'SALARIS OVERZICHT' uit te schrijven, voorafgegaan door tien spaties en gevolgd op de volgende regel door een dubbele onderstreping onder de kop (gebruik het = teken), dus

```
SALARIS OVERZICHT
=================
```

3. De datum moet worden afgedrukt rechts van de woorden SALARIS OVERZICHT in vraag 2, na een hiaat van 20 spaties. Deze datum neemt de vorm dd/mm/jj aan, waarin dd mm en jj dag, maand en jaar (iedere categorie bestaat uit 2 cijfers) voorstellen die door opdrachten uit de Procedure Division een waarde zullen krijgen. De schuine strepen zijn daarentegen constanten die in de kop beschreven moeten worden. Wijzig uw antwoord op vraag 2 om hierin te voorzien (de datum wordt niet onderstreept).

F2 PICTURE-PASSAGE – NUMERIEKE VELDEN

We hebben gezien hoe de picture-passage gebruikt wordt om eenvoudige numerieke, alfabetische en alfanumerieke elementen te beschrijven. We zullen nu onze kennis van numerieke pictures aanvullen en een bijzonder geval van alfanumerieke pictures bekijken - het opgemaakte veld.

Numerieke pictures kunnen behalve het teken 9 bovendien de tekens S, V en P bevatten.

S geeft aan dat het getal een negatieve waarde kan aannemen: het mag een teken hebben. S moet het eerste teken in de picture zijn en het voegt *niets* toe aan de lengte van het veld, tenzij de keuzemogelijkheid SIGN LEADING SEPARATE is toegevoegd. Bijvoorbeeld:

```
02  BELASTING      PICTURE S999.
```

beschrijft een numeriek veld van 3 cijfers dat positief of negatief kan zijn, terwijl

```
02  BELASTING      PICTURE S999 SIGN LEADING SEPARATE.
```

een numeriek veld van 3 cijfers voorafgegaan door een veld van één teken dat een + of een - bevat beschrijft.

V geeft de positie van een decimale punt aan en het voegt ook *niets* toe aan de lengte van het veld. De compiler *neemt aan* dat de decimale punt op de plaats staat die wordt gespecificeerd door de V; een echte decimale punt komt echter niet in de gegevens voor. Dus

```
02  BELASTING      PICTURE S999V99.
```

geeft aan een veld, dat ook negatief mag zijn, met 5 cijfers en een denkbeeldige decimale punt tussen het derde en vierde teken. Zo geeft

```
02  KORTING        PICTURE V99.
```

een veld van 2 cijfers aan met een denkbeeldige decimale punt aan het begin.

P is een schalingsfactor die zelden wordt gebruikt in de praktijk. Hij kan worden gebruikt indien de plaats van de denkbeeldige komma buiten het veld ligt. Ook P voegt *niets* toe aan de lengte van het veld. Voorbeeld:

```
01  MILLISECONDEN PICTURE VPPP999.
```

beschrijft een numeriek veld van 3 cijfers dat rekenkundig wordt behandeld alsof .000 eraan voorafging.

```
01  AANTAL-MILJOENEN PICTURE 999P(6).
```

is een numeriek veld van 3 cijfers dat rekenkundig wordt behandeld alsof er 6 nullen op volgden.

Op de meeste computers is het mogelijk de wijze waarop numerieke gegevens binnen de machine worden opgeslagen te beïnvloeden door een specificatie USAGE COMPUTATIONAL na een numerieke picture-passage. Zo zal bijvoorbeeld 03 NUMERIEK-VELD PICTURE 9(6)V999 USAGE COMPUTATIONAL bij de meeste machines

ervoor zorgen dat NUMERIEK-VELD intern als een binair, in plaats van decimaal, getal wordt opgeslagen. Hierdoor kan een programma, vooral als het veel rekenwerk moet doen, sneller werken. Bovendien kan het resulteren in minder geheugenplaatsruimte voor de opslag van grote getallen. Het werkelijke aantal tekens dat wordt gereserveerd door een picture die is gespecificeerd met USAGE COMPUTATIONAL verschilt van computer tot computer. Raadpleeg voor meer informatie over USAGE en de verwante passage SYNCHRONIZED het handboek van de fabrikant; kennis van deze variaties is niet vereist voor deze tekst.

USAGE COMPUTATIONAL, dat afgekort mag worden als COMP, mag niet samen voorkomen met SIGN LEADING SEPARATE. Deze passages zijn namelijk strijdig met elkaar aangezien COMP de compiler laat weten dat in het betreffende veld uitsluitend getallen, eventueel negatief, zonder aparte tekens terecht zullen komen, terwijl SIGN LEADING SEPARATE aangeeft om het teken juist apart weer te geven. Voor beide passages geldt dat ze op het groepsniveau (level 01) gespecificeerd mogen worden in welk geval alle ondergeschikte elementen opgevat worden alsof voor hen USAGE COMPUTATIONAL of SIGN LEADING SEPARATE is gespecificeerd.

Opgaven

1.
```
01  BOEKHOUD-RECORD SIGN LEADING SEPARATE.
    02 TOTAAL-DEBITEUREN  PICTURE S9(8)V99.
    02 TOTAAL-CREDITEUREN PICTURE S9(8)V99.
    02 AANTAL-DEBITEUREN  PICTURE S9(6).
    02 AANTAL-CREDITEUREN PICTURE S9(6).
```

Hoe lang is het groepsrecord?

2. Wat is er fout aan de volgende codering?

```
01  FAKTUUR COMP.
    02 BRUTO                PICTURE 9(8)V99.
    02 KORTING-PERC         PICTURE V99.
    02 KORTING-BEDRAG       PICTURE 9(8)V99.
  .
  .
  .
   COMPUTE KORTING-BEDRAG ROUNDED = BRUTO * KORTING-PERC * -1.
```

3. Wat is er fout aan het volgende?

```
01  CONTROLE COMP.
    02 VOORRAAD   PICTURE 9(8).
    02 BESTELD    PICTURE 9(8).
    02 OPENSTAAND PICTURE 9(8).
    02 LEVERDATUM PICTURE X(6).
```

F3 DE PICTURE PASSAGE – OPGEMAAKTE VELDEN

Aanvullende tekens voor het opmaken kunnen in een picture-rij worden geplaatst, die de gegevens aanpassen aan de wijze waarop we ze afgedrukt willen hebben.

Tussenzetsels (, (komma) B (spatie (B staat voor blank)) 0 (nul) en / (schuine streep)) kunnen tussengevoegd worden in de picture-rij. Voorbeeld:

```
01 BEDRAG PICTURE 99,999.
```

BEDRAG heeft een komma in de derde tekenpositie. Wanneer we nu 01276 zouden verplaatsen naar BEDRAG, dan zou dat 01,276 bevatten. Merk op dat het tussenzetsel de lengte van het veld *doet toenemen*; BEDRAG is een veld van zes tekens. Bovendien is het *geen* numeriek element. De aanwezigheid van het bijzondere teken houdt in dat dit veld niet mag worden gebruikt in een rekenkundige bewerking (ofschoon het wel het resultaat kan bevatten wanneer de keuzemogelijkheid GIVING wordt gebruikt). *Deze regel is van toepassing op alle opgemaakte elementen.*

Andere voorbeelden: het overplaatsen van 311255 naar een element OPGEMAAKT-GEGEVEN PICTURE 99B99B99 zou het veld van acht tekens dat we OPGEMAAKT-GEGEVEN hebben genoemd een waarde 31_12_55 geven (het onderstreepje stelt hier een spatie voor). De verplaatsing van 1234 naar een element AANTAL-DUIZENDEN PICTURE 9,999,000 geeft aan dit veld met 9 tekens een waarde van 1,234,000.

Vier tussenzetsels (+ (plus) - (minus) $ (dollar) en £ (pond)) kunnen worden geplaatst aan de linkerzijde van een picture-rij. Het dollar teken ($) en het pond teken (£) werken net zo als de zojuist beschreven tekens. Het + teken verandert in een - teken indien het gegeven dat naar het veld is verplaatst negatief is. Het - teken verandert in een *spatie* als het gegeven positief of nul is. Als we dus uitgaan van een element RESULTAAT PICTURE -999,999 en we verplaatsen er een negatieve waarde van 100000 naar toe, dan zal dat element -100,000 bevatten. Het verplaatsen van een positieve waarde van 100000 zal een resultaat _100,000 opleveren. Het plus- en minteken kunnen eveneens worden geplaatst aan de rechterzijde van een picture-rij met hetzelfde effect.

De twee tekens CR of DB kunnen ook aan de rechterzijde van een picture-rij worden geplaatst. Ze hebben tot gevolg dat de letters CR respectievelijk DB worden afgedrukt, als de naar het element gezonden gegevens negatief zijn; anders worden twee spaties afgedrukt.

Een bijzonder tussenzetsel is de decimale punt (.). Indien de picture-rij van een veld een decimale punt insluit, dan zal de denkbeeldige decimale punt in de gegevens in overeenstemming worden gebracht met de feitelijke punt in het veld. Wanneer

gegevens met een waarde 47v45 (vier cijfers) naar een veld ANTWOORD PICTURE 99.99 worden verplaatst, zal ANTWOORD bestaan uit vijf tekens: 47.45. Zodra het ontvangende veld groter is dan het ernaar toegezonden gegevenselement, zullen nullen worden geplaatst in de ongebruikte posities, bijvoorbeeld het zenden van 47v45 naar LANG-ANTWOORD PICTURE 999.999 levert een resultaat 047.450 op.

Onderdrukkende en *drijvende tekens* worden aangewend om de vooropgaande nullen in de gegevens te vervangen door een ander teken. Zodra een significant cijfer wordt tegengekomen in de gegevens, gaan alle onderdrukkende en drijvende tekens aan het werk alsof ze het teken 9 waren.

Het eenvoudigste geval is het teken Z, dat voorafgaande nullen in de gegevens vervangt door spaties. Als een gegeven 001008v89 wordt gezonden naar een veld met PICTURE ZZZ,ZZ9.99, is het resultaat __1,008.89. Indien een gegeven 0v0 naar hetzelfde veld zou worden gezonden, dan is het resultaat ______0.00 (merk op dat het tussenzetsel (de komma) in dit geval wordt behandeld als een Z).

De asterisk * die vaak als bescherming op cheques wordt gebruikt werkt op een soortgelijke wijze. Nu worden de vooropgaande nullen vervangen door asterisks. Het sturen van een gegeven 0017v45 naar een veld met PICTURE*****.99 levert een resultaat ***17.45 op.

Als een van de reeds behandelde tekens + - en $ wordt herhaald, zal het gaan *drijven* naar de linkerzijde van het eerste significante teken in de gegevens. Het zenden van 0012v34 naar een veld met PICTURE $$$9.99 levert dan een resultaat _$12.34 op. Voert men de drijvende symbolen in om nullen te onderdrukken dan moet men niet vergeten een extra symbool links van de picture-rij te plaatsen om rekening te houden met het geval dat het aangevoerde gegeven geen vooropgaande nullen heeft.

Als u wilt dat een veld alleen spaties bevat wanneer er een waarde nul naar wordt toegezonden, kunt u BLANK WHEN ZERO specificeren na een picture-passage of u geeft het veld een picture met louter Z's. Zelfs als een verder geheel uit Z's bestaand element komma's en een decimale punt bevat, zal het spaties afdrukken wanneer er een waarde nul naar wordt toegezonden. Laten we bijvoorbeeld een ontvangend element PICTURE Z,ZZZ.ZZ bekijken; als hiernaar een gegeven met een waarde van v50 wordt gezonden, zal het _____.50 afdrukken; indien een gegeven met een waarde van v05 wordt toegezonden, drukt het _____.05 af en bij toezending van een gegeven met een waarde nul, drukt het acht spaties af.

De opmaak functioneert alleen, wanneer de gegevens intern in het geheugen van de computer worden verplaatst, d.w.z. als resultaat van bijvoorbeeld een rekenkundige bewerking met een GIVING-passage, of als resultaat van een MOVE-instructie (zie paragraaf G4). Gegevens die worden ingelezen vanuit een invoermedium naar een veld met een opmaakpicture worden niet opgemaakt.

In plaats van het dollarteken kunnen we ook het guldenteken (§) gebruiken als we in de ENVIRONMENT DIVISION (zie paragraaf J2) in de SPECIAL-NAMES paragraaf declareren: CURRENCY SIGN IS "§".

Opgaven

1. Het gegeven in het veld waarvan de PICTURE in de linker kolom staat moet worden verplaatst naar een veld met de gegeven picture in de rechter kolom. Schrijf de resulterende inhouden van het ontvangende veld op. Gebruik een onderstreepje om de aanwezigheid van een spatie aan te duiden.

	Picture zendende veld	Gegevens	Picture ontvangende veld	Resultaat
(a)	AAA9	TSR2	AAAB9	
(b)	9(5)	00176	ZZ,ZZ9.99	
(c)	99V9	231	$$$.99	
(d)	9	0	$$$.99	
(e)	S999V99	12645 (positief)	++++9.999	
(f)	999	100	9PP	
(g)	9	2	$$$$$$9.99	

2. De volgende gevallen zijn niet uitdrukkelijk behandeld in de tekst. Schrijf op wat u denkt dat gaat gebeuren.

	Picture zendende veld	Gegevens	Picture ontvangende veld	Resultaat
(a)	9999V99	123456	ZZ9.99	
(b)	999V999	234567	$ZZ9.99	
(c)	99V99	3456	$ZZ9.99	
(d)	S99V99	3456 (negatief)	-ZZ9.99	
(e)	999	100	9PPP	
(f)	V99	01	VPP9	

3. Bekijk de picture-rij: PICTURE $,$$$,$$9.99. Kunt u zeggen wat hier mis aan is (en op sommige compilers ongeoorloofd is)?

F4 PRAKTISCHE OEFENING DEEL 1

Dit is het eerste deel van een praktische oefening die zal worden voltooid wanneer de opdrachten OPEN, CLOSE, READ, WRITE en MOVE zijn behandeld in hoofdstuk G.

Een bestand heeft standaardlabels met als identificatie het woord NUMMERS. Ieder record heeft vier getallen van vier cijfers die zijn geplaatst in de eerste 16 posities, waarbij ieder getal één decimale plaats heeft. De records bestaan uit 80 posities maar de andere posities worden niet gebruikt voor het programma.

Er moet een programma worden geschreven dat ieder record leest, het gemiddelde van de getallen van het record berekent en het resultaat afdrukt samen met de oorspronkelijke getallen. De records moeten worden geteld en het recordnummer moet worden afgedrukt naast de ingevoerde getallen en het gemiddelde. Een kleine complicatie: het vierde getal is altijd negatief ofschoon het niet als zodanig op het record is aangebracht, d.w.z. het moet worden afgetrokken van de som van de andere getallen, voordat het gemiddelde wordt berekend.

De lay-out van het gedrukte rapport ziet er als volgt uit:

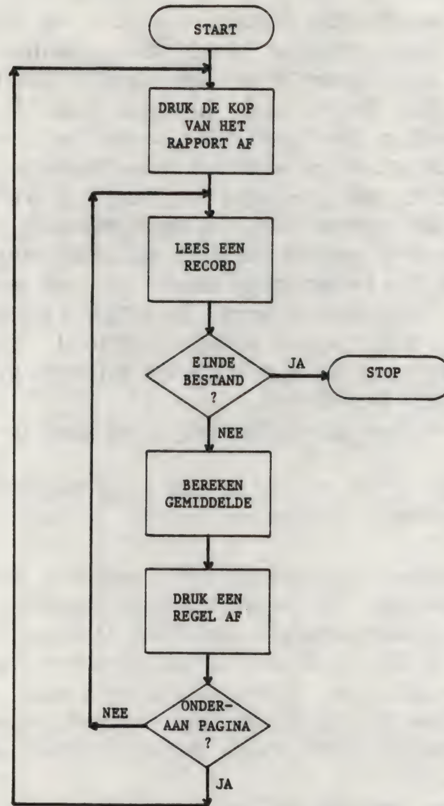
```
RECORDNR    NO 1  NO 2  NO 3  NO 4  GEMIDDELDE
========    ====  ====  ====  ====  ==========
```

Het recordnummer (maximaal 999 records) moet worden afgedrukt in de posities 3 t/m 5. De vier getallen moeten worden afgedrukt in de respectievelijke posities 11 t/m 15, 17 t/m 21, 23 t/m 27 en 29 t/m 33. Het vierde getal moet steeds worden gevolgd door een minteken. Het gemiddelde moet worden afgedrukt in de posities 40 t/m 45, gevolgd door een minteken als het negatief is. Het gemiddelde heeft twee decimale plaatsen.

1. Schrijf een complete Data Division voor dit programma.
2. De Procedure Division kunt u schrijven wanneer hoofdstuk G is behandeld.
3. Als u voorlopig van uw docent de beschrijving van Identification en Environment Division ontvangt kunt u dit programma daarna door de computer laten verwerken.

Een globaal programmastroomschema wordt gegeven op bladzijde 64.

Vergewis u ervan dat de gegevens recht onder de kopjes komen te staan en dat de velden zinvol opgemaakt zijn. Om de overgang op een nieuwe pagina te testen kunt u bijvoorbeeld aannemen dat er 5 regels op een pagina gaan.



FIGUUR 22 Programmastroomschema voor praktische oefening

ANTWOORDEN HOOFDSTUK F

Paragraaf F1

1. De value-passage mag niet worden gebruikt in de File Section.

2.
```
WORKING-STORAGE SECTION.
01  KOP.
    02 FILLER PICTURE X(10) VALUE SPACES.
    02 FILLER PICTURE X(17) VALUE "SALARIS OVERZICHT".
01  ONDERSTREPING.
    02 FILLER PICTURE X(10) VALUE SPACES.
    02 FILLER PICTURE X(17) VALUE "=================".
```

   Een kortere oplossing zou zijn:

```
01  KOP PIC X(27) VALUE "__________SALARIS OVERZICHT",
```

   enzovoort, maar die geeft wel meer aanleiding tot vergissingen en is ook minder gemakkelijk te wijzigen.

3.
```
WORKING-STORAGE SECTION.
01  KOP.
    02 FILLER PICTURE X(10) VALUE SPACES.
    02 FILLER PICTURE X(37) VALUE "SALARIS OVERZICHT".
    02 DAG    PICTURE 99.
    02 FILLER PICTURE X VALUE "/".
    02 MAAND  PICTURE 99.
    02 FILLER PICTURE X VALUE "/".
    02 JAAR   PICTURE 99.
```

ONDERSTREPING is als in 2. Veel andere varianten zijn mogelijk. Ter illustratie is in dit antwoord gebruik gemaakt van de regel dat niet-numerieke literals in een lang veld aan de rechterzijde met spaties worden aangevuld om de extra 20 spaties te verkrijgen. De recht-toe-recht-aan oplossing door het aantal spaties expliciet te vermelden is mogelijk professioneler. Een kortere oplossing is mogelijk door de opmaakfaciliteiten uit paragraaf F3 te gebruiken om de schuine strepen in te voegen, d.w.z. KOP-DATUM PICTURE 99/99/99.

Paragraaf F2

1. 36 tekens.
2. Het resultaat van de compute-opdracht is negatief, maar KORTING-PERC heeft geen PICTURE die met S begint.
3. Een alphanumeriek element (LEVERDATUM) kan niet als USAGE COMPUTATIONAL worden gedeclareerd.

Paragraaf F3

1. (a) TSR 2
   (b) ___176.00
   (c) $23.10
   (d) __$.00
   (e) _+126.4500
   (f) 1
   (g) _____$2.00
2. (a) 234.56
   (b) $234.56
   (c) $_34.56
   (d) -_34.56
   (e) 0
   (f) 0
3. De eerste komma zorgt klaarblijkelijk voor het geval dat de waarde 1 miljoen of meer is. Maar gegevens met een dergelijke waarde die naar het veld met deze picture worden verplaatst hebben als resultaat dat geen enkel $-teken wordt afgedrukt. Er zouden tenminste twee $-tekens voor de komma moeten staan.

G HET WERKEN MET BESTANDEN EN HET TRANSPORT VAN GEGEVENS IN HET GEHEUGEN

G1 OPEN EN CLOSE

Voordat we een record uit een bestand kunnen lezen of een record naar een bestand kunnen wegschrijven, moet het bestand *'open'* zijn, voor respectievelijk invoer of uitvoer. De open-opdracht ziet er als volgt uit:

```
OPEN {INPUT  file-name-1 [file-name-2] ...} ...
     {OUTPUT file-name-3 [file-name-4] ...}
```

Als we dus een programma hebben dat kaarten leest en we ze gebruiken om een stambestand op magneetband bij te werken, waarbij de mutaties op de regeldrukker worden uitgeschreven, dan kunnen we aan het begin van het programma coderen:

```
OPEN INPUT KAARTBESTAND STAM-OUD
     OUTPUT STAM-NIEUW AFDRUKBESTAND.
```

De hier gebruikte bestandsnamen moeten dezelfde zijn als die we hebben gevormd in de FD-opdracht. Het openen heeft bij een bestand met standaardlabels tot gevolg dat we de koplabel controleren (invoer) of uitschrijven (uitvoer). Bij een bestand op magneetschijf zal het labelrecord waarschijnlijk niet aan het begin van het bestand staan, maar ondergebracht zijn in een speciaal gebied met stuurgegevens op de schijf.

Nadat we alle records van het bestand hebben verwerkt, moet het bestand worden *gesloten* (Engels: *closed*). Aan het logische einde van ons programma kunnen we dan coderen:

```
CLOSE STAM-OUD KAARTBESTAND STAM-NIEUW AFDRUKBESTAND.
```

Het afsluiten heeft tot gevolg dat bij uitvoerbestanden op magneetband staartlabels worden uitgeschreven. Maar de exacte

betekenis van CLOSE bij bestanden op andere opslagmedia varieert gewoonlijk enigszins per computermerk. Raadpleeg eveneens voor de exacte betekenis van de varianten CLOSE...WITH NO REWIND en CLOSE...WITH LOCK het handboek van de computerfabrikant. Als u een uitvoerbestand niet afsluit WITH LOCK, gaat minimaal één computer ervan uit, dat u het in het programma gecreëerde bestand slechts tijdelijk nodig heeft: het bestand wordt vernietigd, wanneer het programma is afgelopen.

Soms is het gewenst records toe te voegen aan het einde van een al gecreëerd bestand. Bij de meeste computers kunt u dit zowel bij bestanden op magneetschijf als bij bestanden op magneetband bereiken door te declareren: OPEN EXTEND bestandsnaam.

Opgaven

1. Wanneer u eenmaal een bestand hebt geopend, mag u niet opnieuw een open-opdracht verstrekken tenzij het bestand eerst wordt gesloten. Waarom niet?
2. Wat is het verschil tussen het logische en het fysieke einde van een programma?

G2 READ EN WRITE

De opdracht Read heeft tot gevolg dat het volgende record in het gespecificeerde bestand wordt ingelezen naar het geheugen, in het gebied dat we in de recordbeschrijving voor dat bestand hebben gespecificeerd.

```
READ file-name RECORD AT END imperative-statement.
```

Als we een bestand bijvoorbeeld MUTATIE hebben genoemd en we willen het volgende record krijgen, dan kunnen we coderen READ MUTATIE AT END PERFORM AFSLUITEN. De opdracht achter AT END zal alleen worden uitgevoerd indien de computer bij het lezen van het bestand de staartlabel tegenkomt. Nadat op deze wijze het bestandseinde is ontdekt, zal in het voorbeeld de paragraaf AFSLUITEN worden uitgevoerd.

Wanneer het invoerbestand wordt ingelezen via een apparaat zoals een kaartlezer of een beeldscherm en geen staartlabel heeft, verwacht het bedieningssysteem gewoonlijk een bijzondere boodschap bij de invoer om het bestandseinde te markeren. De aard van deze boodschap varieert per computertype; voorbeelden zijn: een kaart met de tekens /* in de kolommen 1-2, of een regel op het beeldscherm met de tekens ?END. Wanneer deze gegevens worden tegen-

gekomen in de invoer als resultaat van een READ-instructie, wordt de AT END-conditie van kracht.

U kunt een record rechtstreeks inlezen naar een ander recordgebied, dat wordt beschreven in Working Storage, door gebruikmaking van de INTO-mogelijkheid, bijvoorbeeld:

```
READ MUTATIE INTO WERK-MUT AT END PERFORM AFSLUITEN.
```

Het uitschrijven van een record gaat op dezelfde manier, behalve dat we deze keer de *recordnaam* die we willen uitschrijven, moeten specificeren. Bijvoorbeeld:

```
WRITE DISC-REC-1
```

heeft tot gevolg dat het record in de File Section beschreven als DISC-REC-1 wordt uitgeschreven naar het bestand waartoe het behoort. Voor het record dat behoort tot een afdrukbestand, zou WRITE REGEL tot gevolg hebben dat het record wordt uitgevoerd op de afdrukeenheid. In de meeste gevallen zal de afdrukeenheid het papier automatisch een regel opschuiven vóór of na (afhankelijk van de computer) het uitschrijven van de regel. De meeste computers gaan op een nieuwe regel over alvorens hem af te drukken, maar er zijn ook computers die helemaal niet automatisch bij zo'n opdracht op een nieuwe regel overgaan, hetgeen erin resulteert dat bij een WRITE-opdracht de ene regel over de andere wordt afgedrukt.

U kunt een record rechtstreeks uit Working-Storage wegschrijven door het gebruik van de FROM-mogelijkheid, bijvoorbeeld:

```
WRITE AFDRUK-REGEL FROM W-KOPREGEL.
```

Opgaven

1. Schrijf de opdracht om het volgende record uit het bestand INVOER-BAND te lezen. Het record is op het 01-niveau INVOER-REC genoemd. Indien het staartrecord wordt ontmoet, moet u een paragraaf AFSLUITEN genaamd laten uitvoeren en de gang stoppen.
2. U hebt een bestand AFDRUK-BESTAND dat wordt beschreven door en bestaat uit de records REGEL-1 en REGEL-2. Schrijf de opdrachten die tot gevolg hebben dat een record van het type REGEL-1 en een record van het type REGEL-2 worden uitgeschreven op achtereenvolgende regels. (Neem aan dat uw computer automatisch op een nieuwe regel overgaat.)
3. Een bestand kan meer dan één recordtype bevatten, d.w.z. dat twee of meer recordbeschrijvingen volgen op de bestandsbeschrijving. Wanneer u dit weet, kunt u dan verklaren waarom bij READ de bestandsnaam, maar bij WRITE de recordnaam moet worden gespecificeerd?

G3 HET SCHRIJVEN VAN RECORDS NAAR DE REGELDRUKKER

Extra faciliteiten worden gegeven voor het besturen van de interlinie bij het uitschrijven van records naar de regeldrukker. Het formaat is dan:

```
                                                  ⎧identifier LINES⎫
WRITE record-name ⎧AFTER ⎫ ADVANCING ⎨integer LINES   ⎬
                  ⎩BEFORE⎭                        ⎩PAGE            ⎭
```

Verschillende computerfabrikanten hebben òf BEFORE òf AFTER als standaard aangenomen voor hun machines, hoewel de meesten beide mogelijkheden bieden. Ze bieden allemaal de 'integer LINES'-mogelijkheid, waarbij het feitelijke aantal op te schuiven regels wordt gespecificeerd, bijvoorbeeld:

```
WRITE AFDRUK-REGEL BEFORE ADVANCING 10 LINES
```

zal tot gevolg hebben dat een ruimte van 9 regels wordt opengelaten tussen achtereenvolgende regels afgedrukt door deze opdracht.

Wanneer de 'identifier'-mogelijkheid wordt gebruikt, bijvoorbeeld:

```
WRITE AFDRUK-REGEL AFTER INTERLINIE
```

dan zal het papier het aantal regels opschuiven dat wordt aangegeven door het integer gegevenselement dat we INTERLINIE hebben genoemd.

De PAGE-mogelijkheid resulteert in de rechtstreekse opdracht WRITE AFDRUK-REGEL BEFORE ADVANCING PAGE.

Het is een normale vereiste bij het programmeren om het aantal regels, zoals die worden afgedrukt, te tellen en een bepaalde handeling uit te voeren, wanneer de voet van de pagina is bereikt. Dit kan automatisch worden bereikt door het gebruik van de LINAGE-passage in de FD-opdracht van het bewuste bestand, door een specificatie van het aantal regels dat voorafgaat aan de voet van de pagina en door het gebruik van de AT END-OF-PAGE-passage in de WRITE-opdrachten van het bewuste bestand om de gewenste handeling in beweging te zetten. Bijvoorbeeld:

```
FD  AFDRUKBESTAND LABEL RECORDS STANDARD LINAGE 50 LINES.
01  AFDRUKREC PICTURE X(132).
 :
    WRITE AFDRUKREC AFTER ADVANCING 2 LINES
    AT END-OF-PAGE
       PERFORM SCHRIJF-VOET-REGELS
       PERFORM SCHRIJF-PAGINA-KOPREGELS.
```

De instructies na END-OF-PAGE zullen worden uitgevoerd (nadat de WRITE-instructie is gedaan), als de regelteller van de pagina gelijk is aan of groter dan 50 (aantal regels). Het gebruik van WRITE tesamen met het sleutelwoord PAGE (in de paragraaf SCHRIJF-PAGINA-KOPREGELS), zal de regelteller weer op nul zetten.

Opgaven

1. Schrijf de opdracht, die een record FAKTUUR-REGEL afdrukt voordat 2 regels wordt opgeschoven, tenzij het element REGEL-TELLER de waarde 60 bevat, in welk geval FAKTUUR-REGEL afgedrukt moet worden voordat naar de kop van de volgende pagina wordt vooruitgegaan. Het programma moet worden voortgezet bij de paragraaf met als naam: LEES-VOLGEND-RECORD. (Tot deze programmeerstijl worden we nu nog gedwongen indien zoals hier de LINAGE-passage niet beschikbaar is. Verderop, na invoering van een krachtiger PERFORM-opdracht en de invoering van de IF...ELSE-opdracht kunnen we deze lelijke GO TO's vermijden.)
2. Wijzig uw antwoord op vraag 1 zodanig dat een paragraaf AFDRUKKEN-FAKTUUR-KOP wordt uitgevoerd indien een verschuiving naar de kop van de volgende pagina wordt gemaakt.

G4 MOVE

De MOVE-opdracht wordt gebruikt om gegevens binnen het geheugen te verplaatsen. Beschouw het volgende programma:

```
DATA DIVISION.
FILE SECTION.
FD  DISKBESTAND LABEL RECORDS OMITTED.
01  DISKREC PICTURE X(80).
FD  AFDRUKBESTAND LABEL RECORDS OMITTED.
01  REGEL PICTURE X(120).
PROCEDURE DIVISION.
P1-OPENEN.
    OPEN INPUT DISKBESTAND OUTPUT AFDRUKBESTAND.
P2-LEES-EEN-RECORD.
    READ DISKBESTAND AT END GO TO P3-AFSLUITEN.
    MOVE DISKREC TO REGEL.
    WRITE REGEL BEFORE ADVANCING 2 LINES.
    GO TO P2-LEES-EEN-RECORD.
P3-AFSLUITEN
    CLOSE DISKBESTAND AFDRUKBESTAND.
    STOP RUN.
```

Dit programma zal eenvoudig de records van de disk in een niet opgemaakte vorm uitlijsten op de regeldrukker. De MOVE-opdracht wordt gebruikt om de gegevens in het invoerrecordgebied te verplaatsen naar het uitvoerrecordgebied.

Als het wenselijk is de uitvoer op te maken, is het noodzakelijk ieder veld in de invoer en de uitvoer te beschrijven, bijvoorbeeld:

```
01  DISKREC.
    02 EERSTE-BEDRAG   PICTURE 9(4)V99.
    02 TWEEDE-BEDRAG   PICTURE 9(4)V99.
    02 FILLER          PICTURE X(68).
```

en

```
01  REGEL.
    02 BEDRAG-1        PICTURE ZZZ9.99.
    02 FILLER          PICTURE XX.
    02 BEDRAG-2        PICTURE ZZZ9.99.
    02 FILLER          PICTURE X(104).
```

De Procedure Division kan dan worden gewijzigd om te vermelden:

```
P1-OPENEN.
    OPEN INPUT DISKBESTAND OUTPUT AFDRUKBESTAND.
    MOVE SPACES TO REGEL.
P2-LEES-EEN-RECORD.
    READ DISKBESTAND AT END GO TO P3-AFSLUITEN.
    MOVE EERSTE-BEDRAG TO BEDRAG-1.
    MOVE TWEEDE-BEDRAG TO BEDRAG-2.
    WRITE REGEL BEFORE ADVANCING 2 LINES.
    GO TO P2-LEES-EEN-RECORD.
```

De eerste move vult de gehele REGEL met spaties (denk eraan dat we in de File Section geen begin VALUE kunnen geven). De tweede en derde move brengen de individuele velden over en zorgen ervoor dat deze worden opgemaakt in het uitvoerrecord.

De volgende regels zijn van toepassing op gewone moves:

1. *Alfanumeriek element naar alfanumeriek element.*
De gegevens in het veld van verzending worden van links naar rechts naar het ontvangende veld overgebracht. Indien het ontvangende veld te groot is, wordt het aan de rechterkant opgevuld met spaties. Een groepsveld (bijvoorbeeld een record) telt als een alfanumeriek veld.

2. *Numeriek element naar numeriek element.*
De gegevens links van de denkbeeldige decimale punt in het veld van verzending worden ook links van en aansluitend aan de denkbeeldige decimale punt naar het ontvangende veld overgebracht. Voor de komma worden de gegevens van rechts naar links en

achter de komma van links naar rechts overgebracht. Indien het ontvangende veld te groot is, worden kop- of staartnullen ingevoegd al naar wordt vereist.

3. *Numeriek element naar numeriek opgemaakt element.*
Bij het overbrengen komt de denkbeeldige decimale punt in het veld van verzending overeen met de feitelijke decimale punt in het ontvangende veld en de regels voor het opmaken worden opgevolgd.

Het is ook geoorloofd, hoewel zelden de behoefte daaraan bestaat, om een integer numeriek element over te brengen naar een alfanumeriek element. In dit geval worden de gegevens in het veld van verzending van links naar rechts naar het ontvangende veld overgebracht; indien het ontvangende veld te groot is, wordt het aan de rechterkant opgevuld met spaties. Evenzo kan een alfanumeriek element naar een numeriek element worden overgebracht. Het alfanumerieke element wordt verondersteld een geheel getal te bevatten, waarna de regels voor numeriek/numeriek moves worden opgevolgd. Het is de verantwoordelijkheid van de programmeur ervoor te zorgen dat het veld van verzending een geheel getal bevat.

Als het ontvangende veld te klein is, bij welk soort move dan ook, zal het resultaat worden afgekapt.

Opgaven

1.
```
02  ALPHA-1 PICTURE X VALUE "1".
02  ALPHA-2 PICTURE X(4).
```
Wat is de inhoud van ALPHA-2 na: MOVE ALPHA-1 TO ALPHA -2?

2.
```
02  NUM-1   PICTURE 9V9 VALUE 2.3.
02  NUM-2   PICTURE 99V99 VALUE 0.
```
Wat is de inhoud van NUM-2 na: MOVE NUM-1 TO NUM-2?

3.
```
02  NUM-1   PICTURE 99 VALUE 23.
02  ALPHA-2 PICTURE X(4).
```
Wat bevat ALPHA-2 na: MOVE NUM-1 TO ALPHA-2?

G5 MOVE BIJ LITERALS

Het veld van verzending in een move-opdracht kan een literal zijn; in een dergelijk geval worden de normale regels voor move toegepast. Een numerieke literal wordt als een numeriek veld van

verzending gerekend en een niet-numerieke literal als een alfanumeriek veld van verzending.

Wanneer het veld van verzending een figuratieve constante is, wordt het gehele ontvangende veld opgevuld met de gespecificeerde constante. Er is geen verschil of de enkelvoudige of meervoudige versie van de figuratieve constante wordt gebruikt.

D.w.z. MOVE SPACE TO REGEL is synoniem met MOVE SPACES TO REGEL. REGEL zal helemaal worden opgevuld met spaties.

Een gelijksoortig resultaat kunnen we bereiken door literal met ALL te specificeren. Veronderstel dat we een element AVELD PICTURE X(10) hebben. Na MOVE "*" TO AVELD zal AVELD een asterisk gevolgd door negen spaties bevatten. Maar na MOVE ALL "*" TO AVELD zal AVELD geheel gevuld zijn met asterisken. Het sleutelwoord ALL kan ook worden gebruikt in een VALUE-passage, bijvoorbeeld:

```
02  AVELD PICTURE X(10) VALUE ALL "*".
```

heeft hetzelfde resultaat, dus AVELD geheel gevuld met asterisken.

Indien het veld van verzending in een MOVE ALL langer is dan één teken, wordt de reeks tekens herhaald totdat het ontvangende veld is gevuld. Na MOVE ALL "ABC" TO AVELD bevat AVELD ABCABCABCA.

Opgave

1. MOVE ALL HIGH-VALUES TO AVELD zou een geoorloofde opdracht in COBOL zijn, maar de opdracht bevat een overmaat aan woorden. Waarom?

G6 PRAKTISCHE OEFENING DEEL 2

U bent nu in staat de praktische oefening uit paragraaf F3 te voltooien.

Maak testgegevens om uw programma te testen zodra de compilering succesvol is verlopen. Sluit een record met allemaal nullen in, een record met allemaal negens, een record waarop het eerste getal 0001 is en de andere getallen allemaal nul, en een record waarop de eerste drie getallen nul zijn en de vierde een 1.

Voor het testen van de routine m.b.t. de overloopbladen kunt u, i.p.v. een echte grote teststapel te construeren, het aantal regels op 5 per blad houden.

ANTWOORDEN HOOFDSTUK G

Paragraaf G1

1. De koplabel is al ingelezen en gecontroleerd of uitgeschreven.
2. Fysieke einde = laatste gecodeerde opdracht.
   Logische einde = laatste uitgevoerde opdracht.

Paragraaf G2

1. READ INVOER-BAND AT END PERFORM AFSLUITEN STOP RUN.
2. WRITE REGEL-1.
   WRITE REGEL-2.
3. Een bestand kan verschillende soorten records bevatten. Een READ-opdracht heeft tot gevolg, dat het volgende record, van welk type dan ook, op het bestand wordt ingevoerd in het geheugen. Wanneer de READ-opdracht wordt uitgevoerd, is het type van dit volgende record op het bestand niet bekend, zodat het onmogelijk is dit door een recordnaam te specificeren. Daarom staat bij READ de bestandsnaam.
   Omgekeerd: wanneer een WRITE-opdracht wordt uitgevoerd, moet aan de machine worden verteld welke van de verschillende recordtypes die tot een bestand kunnen behoren moet worden uitgeschreven. Dus bij WRITE staat de recordnaam.

Paragraaf G3

1.
```
IF REGELTELLER = 60
   WRITE FAKTUUR-REGEL BEFORE PAGE
   GO TO LEES-VOLGEND-RECORD.
WRITE FAKTUUR-REGEL BEFORE 2.
GO TO LEES-VOLGEND-RECORD.
```

2.
```
IF REGELTELLER = 60
   WRITE FAKTUUR-REGEL BEFORE PAGE
   PERFORM AFDRUKKEN-FAKTUUR-KOP
   GO TO LEES-VOLGEND-RECORD.
WRITE FAKTUUR-REGEL BEFORE 2.
GO TO LEES-VOLGEND-RECORD.
```

Paragraaf G4

1. 1___
2. 0230
3. 23__

Paragraaf G5

1. De ALL is overmatig. MOVE HIGH-VALUES TO AVELD zou hetzelfde resultaat hebben. Figuratieve constanten impliceren MOVE ALL.

H KWALIFICATIE, HERBESCHRIJVING; INDICES

H1 KWALIFICATIE

De namen die we bedenken voor de records in ons programma moeten verschillend zijn. Maar het is geoorloofd dezelfde naam te gebruiken voor velden in verschillende records, bijvoorbeeld:

```
01  DISK-IN.
    02 BEDRAG-1 PICTURE 9(4)V99.
    02 BEDRAG-2 PICTURE 9(4)V99.
    02 FILLER PICTURE X(68).
01  DRUK-AF.
    02 BEDRAG-1 PICTURE ZZZ9.99.
    02 FILLER   PICTURE XX.
    02 BEDRAG-2 PICTURE ZZZ9.99.
    02 FILLER   PICTURE XX.
    02 TOTAAL   PICTURE Z(4)9.99.
```

Een opdracht in de Procedure Division zoals MOVE BEDRAG-1 TO BEDRAG-1 is echter dubbelzinnig en daarom is het in dit geval noodzakelijk verwijzingen naar deze velden te *kwalificeren* in de Procedure Division, bijvoorbeeld:

```
MOVE BEDRAG-1 OF DISK-IN TO BEDRAG-1 OF DRUK-AF.
MOVE BEDRAG-2 IN DISK-IN TO BEDRAG-2 IN DRUK-AF.
ADD BEDRAG-1 OF DISK-IN BEDRAG-2 IN DISK-IN GIVING TOTAAL.
```

Door middel van de sleutelwoorden IN en OF, die synoniemen zijn, wordt een gegevensnaam uniek gemaakt door de naam van het groepsveld waartoe het veld behoort. Indien ook deze kwalificatie nog niet voldoende is, wordt het proces van kwalificatie, het toevoegen van naasthogere groepsvelden, voortgezet totdat de naam uniek is,

Aangezien het gebruik van duplicaatnamen zo kan uitmonden in veel langere opdrachten om de gegevens te verwerken, geven veel programmeurs er de voorkeur aan ze te vermijden.

In de COBOL-standaard van 1974 bestaat een CORRESPONDING-variant van MOVE, ADD en SUBTRACT waardoor kortere instructies mogelijk worden. We zouden bijvoorbeeld in het voorbeeld hierboven in plaats van de eerste twee MOVEs geschreven kunnen hebben:

```
MOVE CORRESPONDING DISK-IN TO DRUK-AF.
```

De compiler zou dan ieder veld in DISK-IN onderzoeken om te zien of er een veld is met dezelfde naam in DRUK-AF; zo ja, dan zou dat veld worden verplaatst. De te verplaatsen velden moeten dezelfde rangnummers hebben en dezelfde groepsvelden (indien aanwezig). Fillers worden genegeerd.

Hoewel het op het eerste gezicht lijkt alsof CORRESPONDING veel coderingswerk kan besparen, is in de praktijk dit effect niet vaak aanwezig. De extra hoeveelheid werk t.b.v. de kwalificatie overtreft gewoonlijk de besparingen als gevolg van het gebruik van CORRESPONDING. Bovendien bestaat er veel minder behoefte aan het gebruik van duplicaatnamen, wanneer de mogelijkheden van de Report Writer (zie hoofdstuk S) beschikbaar zijn en wanneer bestanden ter plaatse worden bijgewerkt i.p.v. ze te kopiëren op een ander medium (Hoofdstuk T). Er is nu voorgesteld de CORRESPONDING-mogelijkheid te laten vervallen.

H2 HERBESCHRIJVING

Een bestand bevat records met de volgende indeling:

tekens	1-6	rekeningnummer
	7-25	naam
	26-85	adres
	86-165	verschepingsadres
	166-169	bestelde hoeveelheid
	170-171	valutacode (01 = Engelse £ sterling 02 = US $ enzovoort)
	172-178	bedrag (3 decimale posities bij £, 2 bij $, geen bij lire enzovoort)

De recordindelingen zijn identiek uitgezonderd het aantal decimale posities in het veld bedrag dat varieert al naar gelang de valutacode. Een oplossing voor dit probleem zou kunnen bestaan in de beschrijving van drie verschillende records voor het bestand, maar deze oplossing is afkeurenswaardig met het oog op de bijna identieke indeling. Een betere oplossing is de beschrijving van het veld bedrag op drie verschillende manieren, d.w.z.:

```
01  FAKTUUR-REC.
    02 REK-NR        PICTURE X9(5).
    02 NAAM          PICTURE X(19).
    02 ADRES         PICTURE X(60).
    02 VERSCH-ADRES  PICTURE X(80).
    02 HOEVEELHEID   PICTURE 9(4).
```

```
02 VALUTA         PICTURE 99.
02 BEDR-STERL     PICTURE 9(4)V999.
02 BEDR-DOLL REDEFINES BEDR-STERL PICTURE 9(5)V99.
02 BEDR-LIRE REDEFINES BEDR-STERL PICTURE 9(7).
```

Indien we in de Procedure Division verwijzen naar BEDR-STERL, zal van dit veld worden aangenomen dat het drie decimale posities bezit; bij BEDR-DOLL twee decimale posities en bij BEDR-LIRE geen decimale plaatsen. Redefines worden dus gebruikt om een alternatieve naam en beschrijving te hechten aan een gegeven veld. Het herbeschrijvende element moet dezelfde veldlengte bezitten als het herbeschreven element.

Groepsvelden kunnen eveneens worden herbeschreven. Laten we bijvoorbeeld veronderstellen dat voor buitenlandse klanten REK-NR hierboven een landcode van twee cijfers bevat, een kredietcode van één cijfer en een serienummer van drie cijfers, terwijl binnenlandse klanten een serienummer hebben van vijf cijfers dat volgt op een speciaal voorvoegsel "I". We kunnen deze variaties beschrijven:

```
02  REK-NR-BUITENLAND.
    03 LAND          PICTURE 99.
    03 KRED-CODE     PICTURE 9.
    03 SER-E         PICTURE 999.
02  REK-NR-BINNENLAND REDEFINES REK-NR-BUITENLAND.
    03 VOORVOEGSEL PICTURE A.
    03 SER-I         PICTURE 9(5).
```

Merk op dat de twee groepen dezelfde lengte hebben (zes tekens). Het zijn twee manieren om dezelfde zes tekens in het geheugen te beschrijven. Redefines mogen niet worden gebruikt op 01-niveau in de File Section (ze zijn hier eigenlijk ook overbodig aangezien van tot hetzelfde bestand behorende records wordt aangenomen dat ze elkaar herbeschrijven in het invoer- of uitvoerrecordgebied).

Opgaven

1. Indien u redefines gebruikt in de Working-Storage Section en u hebt de value-passage gebruikt voor het element dat is herbeschreven, dan mag u niet de value-passage gebruiken voor het element dat herbeschrijft. Waarom niet?

2. Een zescijferig datumveld op een beeldschermregel moet worden beschreven op 02-niveau. De subvelden binnen de datum kunnen òf Engels (DDMMJJ) òf Amerikaans (MMDDJJ) òf Juliaans (JJDDD) zijn. Indien de datum het Juliaanse formaat heeft, wordt het zesde cijfer niet gebruikt. Schrijf dat deel van de Data Division dat het datumveld en zijn subvelden beschrijft.

H3 OCCURS n TIMES

Deze passage kan worden benut om een serie aanliggende velden die dezelfde picture bezitten te beschrijven. Veronderstel dat een record twaalf getallen van vier cijfers bevat, ieder met twee decimale posities. Dit kan worden beschreven:

```
01  VERSCHULDIGDE-BETALINGEN.
    02 BEDRAG OCCURS 12 TIMES PICTURE 99V99.
```

Dit bespaart ons de moeite ieder veld te benoemen. Bij verwijzing naar een specifiek veld moeten we een *index* tussen haken achter de naam van het veld plaatsen. BEDRAG(1) bijvoorbeeld verwijst naar het eerste veld in het record; BEDRAG(2) naar het tweede, enzovoort.

De occurs kan ook op een groepsveld worden toegepast, bijvoorbeeld:

```
01  VERSCHULDIGDE-BETALINGEN.
    02 BETALINGSGROEP OCCURS 12 TIMES.
       03 MAAND   PICTURE 99.
       03 BEDRAG  PICTURE 99V99.
```

Dit is een beschrijving van 12 groepsvelden die ieder een tweecijferige maand en een viercijferig bedrag bevatten. We kunnen de eerste groep bereiken door BETALINGSGROEP(1); het eerste veld in de eerste groep door MAAND(1), enzovoort.

We hoeven geen constante als index te nemen. Het kan ook een variabele zijn met een geheeltallige waarde in de juiste reeks (d.w.z. in de reeks 1 t/m 12 in het voorbeeld). Dit levert ons een krachtig mechanisme op. Veronderstel dat we in het voorbeeld hierboven alle bedragen willen totaliseren. We zouden dan kunnen schrijven ADD BEDRAG(1) BEDRAG(2) BEDRAG(3) enzovoort, maar dit zou bijzonder langdradig worden. Door de beschrijving van twee elementen in working storage:

```
02  SUB    PICTURE 99 COMP.
02  TOTAAL PICTURE 9(4)V99 COMP VALUE 0.
```

kunnen we alle bedragen optellen met de volgende opdrachten in de Procedure Division:

```
    MOVE 1 TO SUB.
SOM-BEDR.
    ADD BEDRAG(SUB) TO TOTAAL ON SIZE ERROR (neem nodige
                                              maatregelen).
    ADD 1 TO SUB.
    IF SUB LESS THAN 13
       GO TO SOM-BEDR.
```

Over het algemeen is het uit efficiency overwegingen aan te bevelen om gegevenselementen die als indices gebruikt worden met de juiste USAGE te beschrijven. Op de meeste computers is dit COMP of COMP SYNC RIGHT.

Het is niet geoorloofd een value-passage te gebruiken bij een element of groep die in een OCCURS voorkomt. Indien we toch een value willen invoegen, dan kan dit worden bereikt door herbeschrijving, bijvoorbeeld:

```
01  TABEL.
    02 TABEL-INHOUD.
       03 FILLER PICTURE XXX VALUE "JAN".
       03 FILLER PICTURE XXX VALUE "FEB".
       03 FILLER PICTURE XXX VALUE "MRT".
       03 FILLER PICTURE XXX VALUE "APR".
       03 FILLER PICTURE XXX VALUE "MEI".
       03 FILLER PICTURE XXX VALUE "JUN".
       03 FILLER PICTURE XXX VALUE "JUL".
       03 FILLER PICTURE XXX VALUE "AUG".
       03 FILLER PICTURE XXX VALUE "SEP".
       03 FILLER PICTURE XXX VALUE "OKT".
       03 FILLER PICTURE XXX VALUE "NOV".
       03 FILLER PICTURE XXX VALUE "DEC".
    02 TABEL-2 REDEFINES TABEL-INHOUD.
       03 ALFA-MAAND OCCURS 12 TIMES PICTURE XXX.
```

ALFA-MAAND(1) bevat daardoor de tekens JAN, ALFA-MAAND(2) de tekens FEB, enzovoort. Ik heb al deze fillers gebruikt terwille van de duidelijkheid. Omdat we hier te maken hebben met niet-numerieke waarden kan precies hetzelfde resultaat worden bereikt door:

```
01  TABEL VALUE "JANFEBMRTAPRMEIJUNJULAUGSEPOKTNOVDEC".
    02 ALFA-MAAND OCCURS 12 TIMES PICTURES XXX.
```

Er is enige reden om de eenvoudige, maar lange methode professioneler te noemen, aangezien het programma gemakkelijker kan worden gewijzigd; maar misschien is dat niet zo belangrijk bij een zeer vaste verzameling gegevens zoals de namen van de maanden. Als de tabelinhoud aanpassingen zou vereisen, verdient het de voorkeur de tabelgegevens in een bestand vast te leggen en deze gegevens in de tabel in te lezen bij de start van het programma.

Wanneer een waarde is gedeclareerd voor een groepselement, mag er geen enkele tegenstrijdige declaratie (d.w.z. een andere waarde) bestaan voor welk ondergeschikt element van de groep dan ook.

OCCURS mag niet worden gebruikt op het 01-niveau.

Opgave

1. De File Section van een programma bevat een invoerrecord in BLDSCHBESTAND en een uitvoerrecord in AFDRUKBESTAND als volgt:

```
01  BEELDSCHERM-IN.
    02 DATUM-IN.
       03 DAG-IN    PICTURE 99.
       03 MAAND-IN  PICTURE 99.
       03 JAAR-IN   PICTURE 99.
01  REGEL-UIT.
    02 FILLER       PICTURE X(100).
    02 DAG-UIT      PICTURE Z9BB.
    02 MAAND-UIT    PICTURE XXXBB.
    02 JAAR-UIT     PICTURE 99.
```

Schrijf de Working-Storage Section en de Procedure Division voor het inlezen van één regel en het afdrukken van de datum op de regeldrukker. MAAND-UIT moet een uit drie tekens bestaand equivalent van MAAND-IN zijn en kan worden gevonden door MAAND-IN als de index van een element in een tabel te gebruiken.

ANTWOORDEN HOOFDSTUK H

Paragraaf H2

1. U zou dan de compiler vragen om twee verschillende values in hetzelfde geheugengebied te plaatsen, hetgeen onmogelijk is.

2.
```
02 E-DATUM.
   03 E-DAG     PICTURE 99.
   03 E-MAAND   PICTURE 99.
   03 E-JAAR    PICTURE 99.
02 A-DATUM REDEFINES E-DATUM.
   03 A-MAAND   PICTURE 99.
   03 A-DAG     PICTURE 99.
   03 A-JAAR    PICTURE 99.
02 J-DATUM REDEFINES E-DATUM.
   03 J-JAAR    PICTURE 99.
   03 J-DAG     PICTURE 999.
   03 FILLER    PICTURE 9.
```

Paragraaf H3

1.
```
WORKING-STORAGE SECTION.
01  TABEL VALUE  "JANFEBMRTAPRMEIJUNJULAUGSEPOKTNOVDEC".
    02 ALFA-MAAND OCCURS 12 TIMES PICTURE XXX.
01  INLEZEN PICTURE X(4) VALUE "GOED".
PROCEDURE DIVISION.
P-HOOFDPROGRAMMA.
    PERFORM P1-OPENEN.
    PERFORM P2-LEZEN.
    IF INLEZEN = "GOED"
       PERFORM P3-AFDRUKKEN.
    PERFORM P4-AFSLUITEN.
    STOP RUN.
P1-OPENEN.
    OPEN INPUT BLDSCHBESTAND OUTPUT AFDRUKBESTAND.
P2-LEZEN.
    READ BLDSCHBESTAND AT END PERFORM P21-FOUTMELDING.
P21-FOUTMELDING.
    DISPLAY "GEEN RECORDS IN BLDSCHBESTAND-JOB MISLUKT".
    MOVE "FOUT" TO INLEZEN.
P3-AFDRUKKEN.
    MOVE SPACES TO REGEL-UIT.
    MOVE DAG-IN TO DAG-UIT.
    MOVE ALFA-MAAND(MAAND-IN) TO MAAND-UIT.
    MOVE JAAR-IN TO JAAR-UIT.
    WRITE REGEL-UIT.
P4-AFSLUITEN.
    CLOSE BLDSCHBESTAND AFDRUKBESTAND.
```

I PROCEDURE DIVISION (VERVOLG)

I1 VOORWAARDELIJKE UITDRUKKINGEN

Deze hebben het formaat:

```
IF {simple-condition  } {statement-1  } [ELSE statement-2]
   {compound-condition} {NEXT SENTENCE}
```

We hebben dit al in gebruik gezien waar de enkelvoudige voorwaarde (simple-condition) bestaat uit de gelijkheid van twee elementen (D1), bijvoorbeeld:

```
IF  RECORD-CODE = 1
    PERFORM BEREKENING-1.
```

De ELSE-variant stelt ons in staat een opdracht te schrijven die alleen wordt uitgevoerd als aan de voorwaarde niet wordt voldaan, bijvoorbeeld:

```
IF  RECORD-CODE = 1
    PERFORM BEREKENING-1
ELSE
    PERFORM BEREKENING-OVERIGE.
```

Andere enkelvoudige voorwaarden die kunnen worden gespecificeerd staan hieronder samengevat:

```
                             {GREATER THAN}
                             {LESS THAN   }
{identifier-1}  IS [NOT]     {EQUAL TO    }  {identifier-2}
{literal-1   }               {=           }  {literal-2   }
                             {>           }
                             {<           }
```

Waar beide met elkaar vergeleken velden numerieke elementen zijn, wordt een algebraïsche vergelijking gemaakt (d.w.z. positieve getallen zijn groter dan 0 en negatieve getallen zijn kleiner dan 0,

en ook is bijvoorbeeld -7 groter dan -8). In andere gevallen worden de velden per teken van links naar rechts met elkaar vergeleken, waarbij het kortste veld wordt behandeld alsof het spaties heeft aan de rechterkant. Hoewel alle computerfabrikanten Z als groter dan A en 9 groter dan 0 behandelen, treden variaties in de praktijk op wanneer een letter wordt vergeleken met een getal of wanneer bijzondere tekens in de vergelijking betrokken zijn.

In ASCII (American Standard Code for Information Interchange, die tevens de ISO 7-bit code en de British Standard Data Code is) zijn cijfers kleiner dan hoofdletters, die op hun beurt weer kleiner zijn dan onderkast (= kleine) letters. In EBCDIC (Extended Binary Coded Decimal Interchange Characters), zijn cijfers groter dan hoofdletters, die op hun beurt groter zijn dan onderkast letters. De wiskundige symbolen > en < kunnen worden gebruikt in plaats van GREATER THAN en LESS THAN.

Velden kunnen ook naar type worden getoetst:

```
                            POSITIVE
                            NEGATIVE
identifier IS [NOT]         ZERO
                            NUMERIC
                            ALPHABETIC
```

De picture van het element behoort overeen te komen met de gemaakte toets; in het bijzonder mag een NUMERIC test niet worden uitgevoerd op een element met een alfabetische picture, noch een ALPHABETIC test op een element met een numerieke picture.

Opgave

1. Een rekeningnummer heeft zes tekens. Het eerste is altijd de letter A, het volgende teken kan iedere vorm aannemen en de laatste vier tekens behoren een getal in de reeks 0001 t/m 5000 te zijn. Beschrijf dit veld in de Data Division en schrijf de opdrachten van de Procedure Division die controleren of de gegevens in dat veld overeenstemmen met deze regels. Als het gegeven niet correct is moet de procedure FOUTMELDING worden uitgevoerd.

12 SAMENGESTELDE VOORWAARDEN

Enkelvoudige voorwaarden kunnen door de woorden AND en OR worden samengevoegd om een samengestelde voorwaarde te maken. We kunnen bijvoorbeeld coderen:

```
IF EERSTE-TEKEN = "A" AND LAATSTE-VIER GREATER THAN 0
   NEXT SENTENCE
ELSE PERFORM FOUTMELDING.
```

Aan beide enkelvoudige voorwaarden moet zijn voldaan wil het programma met de volgende zin doorgaan; FOUTMELDING zal dus al worden uitgevoerd wanneer slechts één van beide condities onwaar is. Als OR wordt gebruikt, zal de voorwaardelijke opdracht worden uitgevoerd indien aan één van beide voorwaarden wordt voldaan; de logica van het voorafgaande voorbeeld kan dus ook worden uitgedrukt door:

```
IF EERSTE-TEKEN NOT = "A" OR LAATSTE-VIER LESS THAN 1
PERFORM FOUTMELDING.
```

Wanneer AND en OR allebei worden gebruikt in een samengestelde voorwaarde, dan is het resultaat een zin die in alledaags taalgebruik dubbelzinnig zou zijn. Hoewel COBOL dit oplost door eerst alle AND's op hun waarde te beoordelen, wordt de student toch aangeraden altijd zijn bedoeling te verduidelijken door het gebruik van haakjes, bijvoorbeeld:

```
IF EERSTE-TEKEN NOT = "A" OR
                    (LAATSTE-VIER NOT GREATER THAN ZERO AND
                    LAATSTE-VIER NOT LESS THAN 5001)
   PERFORM FOUTMELDING.
```

Wanneer haakjes worden gebruikt wordt altijd eerst nagegaan of aan de uitdrukking tussen de haakjes is voldaan. Als we nauwkeurig de uitdrukking tussen haakjes in het vorige voorbeeld bekijken, dan zien we dat hier sprake is van een fout (altijd onwaar). NOT GREATER THAN is namelijk hetzelfde als 'kleiner dan of gelijk aan' en NOT LESS THAN is hetzelfde als 'groter dan of gelijk aan'. Het is heel gemakkelijk zowel voor een beginnende als een ervaren programmeur samengestelde voorwaarden te laten uitdijen tot kromme en kronkelige denkoefeningen. Het is bijna altijd mogelijk een samengestelde voorwaarde te vereenvoudigen door NOT's te elimineren en/of de voorwaarden in afzonderlijke zinnen op te schrijven. Soms brengt dit met zich mee dat de zich vertakkende logica wordt omgedraaid; het hierboven genoemde kan worden herschreven, maar nu goed:

```
IF EERSTE-TEKEN NOT = "A" OR LAATSTE-VIER LESS THAN 1
                          OR LAATSTE-VIER GREATER THAN 5000
   PERFORM FOUTMELDING.
```

De meeste ervaren programmeurs hangen het standpunt aan, dat terwille van de eenvoud AND's en OR's niet behoren voor te komen in dezelfde samengestelde voorwaarde. Velen vinden bovendien dat samengestelde voorwaarden in hun geheel moeten worden vermeden.

Opgaven

1. Welke logische tegenspraak staat er tussen de haakjes in het voorbeeld op bladzijde 85?
2. Schrijf opdrachten voor de Procedure Division die de volgende logica uitdrukken:
   a) Indien KLANT-CODE staat in de reeks A t/m D, dan is de KORTING 0.10. Voor andere waarden van KLANT-CODE is de KORTING nul.
   b) Indien VERKOOP-DAG niet-numeriek is of VERKOOP-MAAND niet valt in de reeks 1 t/m 12, moet het programma de procedure FOUTMELDING uitvoeren.

I3 GENESTE IF-OPDRACHTEN

Het is volkomen geoorloofd een IF-opdracht in te bedden in een andere IF-opdracht. Alleen als dit nesten het gevolg is van een grondige en goed gedocumenteerde analyse van de logica, zoals beschreven in deel 2 kan het worden toegepast, anders moet het worden vermeden omdat het kan leiden tot nodeloos ingewikkelde programma's.

Er zijn twee hoofdvarianten. De eerste luidt:

```
IF A = B IF C = D PERFORM P1 ELSE PERFORM P2
ELSE PERFORM P3.
```

Dit betekent: PERFORM P1, indien zowel A = B als C = D; PERFORM P2, indien A = B maar C ≠ D; PERFORM P3, indien A ≠ B. De regel is dat de binnenste ELSE wordt gepaard aan de binnenste IF, de volgende ELSE aan de eerdere IF en zo naar buiten toe door te werken. Als u de verleiding om geneste IF's te gebruiken niet kunt weerstaan, kunt u uzelf en andere lezers van uw programma helpen door iedere IF-opdracht te laten inspringen en de overeenkomstige ELSE-opdrachten daaronder te coderen, d.w.z.:

```
IF  A = B
    IF C = D
       PERFORM P1
    ELSE
       PERFORM P2
ELSE
    PERFORM P3.
```

De andere hoofdvariant is als volgt:

```
IF  E = F PERFORM P4 ELSE IF G = H PERFORM P5
ELSE PERFORM P6.
```

Dit betekent: indien E = F, PERFORM P4; indien E ≠ F, maar G = H, PERFORM P5; indien E ≠ F en G ≠ H, PERFORM P6. Het behoort te worden gecodeerd:

```
IF  E = F
    PERFORM P4
ELSE
    IF G = H
       PERFORM P5
    ELSE
       PERFORM P6.
```

Er zijn geen opgaven bij deze paragraaf.

I4 VARIANTEN VAN DE PERFORM-OPDRACHT

a) PERFORM procedure-name $\left\{\begin{matrix}\text{identifier}\\ \text{integer}\end{matrix}\right\}$ TIMES

Dit zal tot gevolg hebben dat de gespecificeerde paragra(a)f(en) het aantal keren dat is aangegeven wordt (worden) uitgevoerd. Om bijvoorbeeld de som van de waarden van een aantal velden beschreven in een occurs-passage te vinden:

```
02 BEDR-1 OCCURS 100 TIMES PICTURE 9(4)V99.
02 ANTWOORD                PICTURE 9(6)V99.
02 SUB                     PICTURE 999 COMP.
```

kunnen we coderen:

```
        MOVE ZERO TO ANTWOORD.
        MOVE 1 TO SUB.
        PERFORM TEL-OP 100 TIMES.
        PERFORM AFDRUK-ANTWOORD.
        STOP RUN.
TEL-OP.
        ADD BEDR-1(SUB) TO ANTWOORD.
        ADD 1 TO SUB.
```

b) PERFORM procedure-name UNTIL condition

Dit heeft tot gevolg dat de gespecificeerde paragra(a)f(en) wordt (worden) uitgevoerd totdat aan de voorwaarde wordt voldaan. De voorwaarde kan enkelvoudig of samengesteld zijn. De paragraaf

wordt *geen enkele keer* uitgevoerd wanneer *direct* aan de voorwaarde wordt voldaan. Om het laatste voorbeeld te herhalen en uit te breiden:

```
       02 ANTWOORD              PICTURE 9(5)V99.
       .
       .
       .
       MOVE ZERO TO ANTWOORD.
       MOVE 1 TO SUB.
       PERFORM TEL-OP UNTIL SUB = 101 OR FOUT-SEIN = "J".
       PERFORM AFDRUK-ANTWOORD.
       STOP RUN.
   TEL-OP.
       ADD BEDR-1(SUB) TO ANTWOORD  ON SIZE ERROR
                 MOVE "J" TO FOUT-SEIN
                 MOVE ZEROES TO ANTWOORD.
       ADD 1 TO SUB.
```

Er wordt op de voorwaarde getest, voordat de paragraaf wordt uitgevoerd, dus als de voorwaarde waar is aan het begin, wordt de paragraaf geen enkele maal uitgevoerd.

Deze variant van de perform-opdracht kan als de krachtigste en meest universele worden beschouwd. Want hoewel de perform... varying-variant (zie hieronder) eveneens gebruikt zou kunnen worden als het aantal malen dat de procedure uitgevoerd moet worden onbekend is, worden deze laatste variant en ook de voorgaande variant (perform ... times) gewoonlijk gebruikt als dat aantal wel bekend is.

c) PERFORM...VARYING...FROM...BY...UNTIL condition

Dit is een handige vorm van de perform-opdracht die ons in staat stelt om de initiële waarde en stapgrootte met betrekking tot de uitvoering van de out-of-line paragraaf vast te stellen. In het voorgaande voorbeeld kunnen we dan de value-passage voor SUB laten wegvallen en coderen:

```
       PERFORM TEL-OP VARYING SUB FROM 1 BY 1 UNTIL
               SUB = 101 OR FOUT-SEIN = "J".
       PERFORM AFDRUK-ANTWOORD.
       STOP RUN.
   TEL-OP.
       ADD BEDR-1(SUB) TO ANTWOORD  ON SIZE ERROR
                  MOVE "J" TO FOUT-SEIN
                  MOVE ZEROES TO ANTWOORD.
```

Opgaven

1. In het Ruritanische belastingstelsel krijgen de werknemers een codering in de reeks A t/m G die volgens onderstaande tabel de belasting aangeeft die van het salaris moet worden ingehouden.

Belasting code	Belasting
A	0
B	100
C	200
D	350
E	550
F	800
G	1100

Deze twee tabellen worden in de working storage beschreven als volgt:

```
01  TABELLEN.
    02 TAB-VAN-CODES VALUE "ABCDEFG".
       03 BEL-CODE OCCURS 7 TIMES PICTURE X.
    02 TAB-INH VALUE "0000010000200035005500800 1100".
       03 BELASTING OCCURS 7 TIMES PICTURE 9(4).
```

Een werknemersrecord wordt door het programma ingelezen. Het bevat een picture-veld, WRKN-BEL-CODE genaamd, waarvan gecontroleerd is dat het in de reeks A t/m G ligt. Schrijf opdrachten die de juiste BELASTING van BELASTBAAR-INKOMEN aftrekken. Het programma moet vervolgen met BEREKEN-NETTO-SALARIS.

2. In working storage worden de volgende elementen beschreven:

```
01  CONTROLE-ELEMENTEN COMP.
    02 TEMP    PICTURE 999.
    02 FACTOR  PICTURE 9.
    02 SUB     PICTURE 9.
    02 SOM     PICTURE 999.
01  REK-NO.
    02 CIJFER OCCURS 6 TIMES PICTURE 9.
    02 CONTROLE-CIJFER PICTURE 9.
```

Ieder cijfer in REK-NO moet worden vermenigvuldigd met een FACTOR, hetgeen een produktcijfer oplevert. De factor is 7 voor het eerste (linker)cijfer, 6 voor het tweede cijfer en zo naar beneden tot en met 2 voor het laatste cijfer. De som van de produktcijfers moet door 10 worden gedeeld en de rest, afgetrokken van 10, moet in CONTROLE-CIJFER worden geplaatst. Codeer dit.

ANTWOORDEN HOOFDSTUK I

Paragraaf I1

1.
```
01  REK-NO.
    02 EERSTE-TEKEN  PICTURE X.
    02 TWEEDE-TEKEN  PICTURE X.
    02 LAATSTE-VIER  PICTURE 9(4).
PROCEDURE DIVISION.
    }
    IF EERSTE-TEKEN NOT = "A" PERFORM FOUTMELDING.
    IF LAATSTE-VIER NOT NUMERIC PERFORM FOUTMELDING.
    IF LAATSTE-VIER LESS THAN 1 PERFORM FOUTMELDING.
    IF LAATSTE-VIER GREATER THAN 5000 PERFORM FOUTMELDING.
```

Als u wilt dat FOUTMELDING slechts eenmaal wordt uitgevoerd ook al bevat het REK-NO meer dan één soort fout, dan kan dit worden bereikt door middel van een samengestelde voorwaarde (zie volgende paragraaf) of door achter iedere PERFORM (voor de afsluitende punt) een GO TO-opdracht te plaatsen die over de overige testvragen heenspringt.

Paragraaf I2

1. Een getal kan niet zowel kleiner dan of gelijk aan 0 zijn èn tevens groter dan of gelijk aan 5001.

2. a)
```
IF KLANT-CODE LESS THAN "A" OR
    KLANT-CODE GREATER THAN "D"
    MOVE ZERO TO KORTING
ELSE MOVE .1 TO KORTING.
```

   b)
```
IF VERKOOP-DAG NOT NUMERIC OR
    VERKOOP-MAAND LESS THAN 1 OR
    VERKOOP-MAAND GREATER THAN 12
    PERFORM FOUTMELDING.
```

Paragraaf I4

1. Vele antwoorden zijn mogelijk. Een eenvoudig antwoord is dit:

```
    MOVE 1 TO SUB.
    PERFORM VERMEERDERING-SUB
        UNTIL WRKN-BEL-CODE = BEL-CODE(SUB).
    SUBTRACT BELASTING(SUB) FROM BELASTBAAR-INKOMEN.
BEREKEN-NETTO-SALARIS.
{
VERMEERDERING-SUB.
    ADD 1 TO SUB.
```

2. Opnieuw zijn vele varianten mogelijk. Voorbeeld:

```
    MOVE 0 TO SOM.
    PERFORM BEREKEN-EN-TEL-OP VARYING SUB FROM 1 BY 1
                              UNTIL SUB = 7.
    DIVIDE 10 INTO SOM GIVING SOM REMAINDER CONTROLE-CIJFER.
    SUBTRACT CONTROLE-CIJFER FROM 10 GIVING CONTROLE-CIJFER.
    {
    {
BEREKEN-EN-TEL-OP.
    SUBTRACT SUB FROM 8 GIVING FACTOR.
    MULTIPLY FACTOR BY CIJFER(SUB) GIVING TEMP.
    ADD TEMP TO SOM.
```

J DE IDENTIFICATION EN ENVIRONMENT DIVISIONS

J1 DE IDENTIFICATION DIVISION

Deze bestaat uit de volgende paragrafen:

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. program-name.
[AUTHOR. comment.]
[INSTALLATION. comment.]
[DATE-WRITTEN. comment.]
[DATE-COMPILED.]
[SECURITY. comment.]
```

Alleen PROGRAM-ID is verplicht; de programmanaam moet overeenkomen met de regels voor de specifieke computer die men gebruikt.

De resterende paragrafen worden niet gebruikt door de compiler, behalve dat de compilatie-datum aan de paragraaf-naam DATE-COMPILED wordt toegevoegd. Het nut hiervan bij moderne computers is twijfelachtig omdat bijna alle compilers de datum toch al automatisch op de uitlijsting van het programma zullen plaatsen. Dit is gemakkelijk bij het nagaan welke uitlijsting van een programma dat we bezig zijn te ontwikkelen de laatste is.

De andere paragrafen spreken voor zichzelf. Ieder commentaar kan op iedere plaats in het tekstgebied worden geschreven, mits een asterisk wordt geplaatst in kolom 7 van iedere regel (bij compilers met niet voorgeschreven lay-out aan het begin van de regel). Dit maakt de voorziening van speciale paragrafen t.b.v. commentaar enigszins overbodig en daarom is er nu voorgesteld deze opdrachten uit de standaard te halen. Toch is het nog steeds een goede gewoonte om deze paragrafen te gebruiken als een controlelijst van commentaren die aan het begin van het programma gemaakt moeten worden.

Het kan bijzonder nuttig zijn de doeleinden aan het einde van de Identification Division samen te vatten in gewoon Nederlands:

```
*SAMENVATTING
*    INVOER     -STAMBESTAND KLANTEN OUD
*               -GECONTROLEERDE MUTATIERECORDS
*    UITVOER    -STAMBESTAND KLANTEN NIEUW
*               -FOUTENOVERZICHT VAN MUTATIERECORDS (WIJZI-
*                GINGEN EN VERWIJDERINGEN) WAARVOOR GEEN
*                CORRESPONDEREND STAMRECORD BESTAAT, EN VAN
*                IN TE VOEGEN MUTATIERECORDS WAAR EEN STAM-
*                RECORD AL BESTAAT
*    VERWERKING OVEREENKOMSTIG HET MUTATIECODE VELD ALS VOLGT
*                I (INVOEGEN) HET MUTATIERECORD WORDT NAAR HET
*                  NIEUWE STAMBESTAND UITGESCHREVEN
*                W (WIJZIGEN) DE NIET-LEGE VELDEN VAN HET
*                  MUTATIERECORD WORDEN VERPLAATST NAAR DE
*                  JUISTE PLAATSEN VAN HET STAMRECORD
*                V (VERWIJDEREN) HET STAMRECORD WORDT NIET
*                   NAAR HET NIEUWE STAMBESTAND UITGESCHREVEN.
```

Opgave

1. Schrijf de Identification Division voor een programma dat een bestand met records van 80 tekens leest, ze in een opklimmende reeks op grond van de eerste zes kolommen sorteert en ze uitlijst op de regeldrukker. De installatie is Computer Service Centrale en er is geen security-belang voor het programma.

J2 DE ENVIRONMENT DIVISION

De division bestaat uit twee sections, de Configuration Section en de Input-Output Section. De beschrijvingen in deze sections variëren aanzienlijk van fabrikant tot fabrikant en bovendien zullen ze bij een bepaald computersysteem tot op zekere hoogte afhankelijk zijn van de precieze aard van de technische uitrusting die bij dat computersysteem wordt gebruikt. De nu volgende beschrijving licht slechts dat deel van de Environment Division toe dat de meeste fabrikanten en installaties gemeen hebben.

De Configuration Section heeft drie fundamentele paragrafen:

```
CONFIGURATION SECTION.
SOURCE-COMPUTER. computer-name.
OBJECT-COMPUTER. computer-name.
[SPECIAL-NAMES. entry.]
```

Bij source-computer en object-computer plaatst men de namen van de computers die worden gebruikt voor respectievelijk het compileren en het uitvoeren. Gewoonlijk is dit dezelfde computer. Bij bepaalde compilers kan de paragraaf object-computer worden uitgebreid om veel andere kenmerken van de computer vast te leggen. Deze beide paragrafen zijn nu nog verplicht in standaard COBOL, maar er is voorgesteld om ze facultatief te maken.

De facultatieve paragraaf Special-Names kan worden gebruikt om de stand van bepaalde schakelaars van de computer te weten te komen; om een codeverzameling aan te geven zodat bestanden in een andere code gelezen of weggeschreven kunnen worden dan de bij de computer behorende; om aan te geven dat van een ander valutateken gebruik moet worden gemaakt dan het $-teken of om aan te geven dat, zoals in Europa gebruikelijk is, de conventies voor de decimale punt en komma moeten worden aangehouden (die conventies zijn precies omgekeerd ten opzichte van Amerika en Engeland). De laatste twee mogelijkheden stellen ons in Nederland in staat om met het $-teken te werken en op de gebruikelijke manier in getallen met komma en punt om te gaan. De codering luidt:

```
SPECIAL-NAMES.
    CURRENCY SIGN IS "F".
    DECIMAL-POINT IS COMMA.
```

Alle regels met betrekking tot het dollarteken in de PICTURE beschrijving (paragraaf F3) gelden nu voor het nieuwe valutateken, terwijl het dollarteken in dit programma niet meer mag worden gebruikt als valutateken.

Wanneer men DECIMAL-POINT IS COMMA heeft gespecificeerd, verwisselen punt en komma van rol, zowel in de PICTURE-beschrijvingen als in het programma. Voorbeeld (met DECIMAL-POINT IS COMMA):

```
        :
        02 BEDRAG PICTURE 9.999,99.
        :
        :
        MOVE 3128,64 TO BEDRAG.
```

Zonder DECIMAL-POINT IS COMMA had dit moeten luiden:

```
.
.
02 BEDRAG PICTURE 9,999.99.
.
.
.
.
MOVE 3128.64 TO BEDRAG.
```

De meeste computersystemen hebben een set met gestandaardiseerde beschrijvingen voor deze Section, die eenvoudig kunnen worden gekopieerd bij ieder programma.

De Input-Output Section heeft twee paragrafen waarvan een eenvoudige beschrijving als volgt luidt:

```
INPUT-OUTPUT SECTION.
FILE-CONTROL. {SELECT file-name ASSIGN TO device-name
               [RESERVE integer AREAS]} ...
[I-O-CONTROL. SAME RECORD AREA FOR file-name-1, file-name-2.]
```

De paragraaf File-Control wordt gebruikt om het bestand dat we hebben beschreven in de File Section van de Data Division te koppelen aan een bepaald randapparaat. De te gebruiken naam voor dit randapparaat verschilt per computer. Veronderstel dat we een machine gebruiken die de apparaatnamen BEELDSCHERM en DRUKKER toestaat; we kunnen dan coderen:

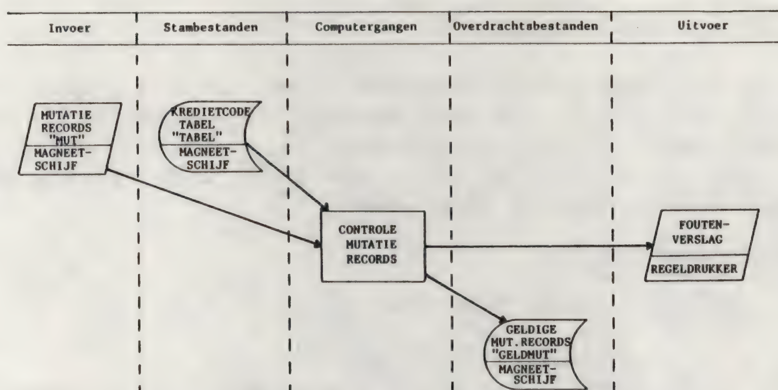
```
FILE-CONTROL. SELECT BEELDSCHBESTAND ASSIGN TO BEELDSCHERM.
              SELECT RAPPORTBESTAND ASSIGN TO DRUKKER.
```

Op bepaalde machines wordt het apparaat niet expliciet genoemd, maar wordt het bestand in plaats daarvan gekoppeld aan een symbolische naam. Deze symbolische naam wordt dan op zijn beurt door opdrachten aan het operating system weer gekoppeld aan een echt apparaat.

Ieder bestand heeft een bepaald deel van het geheugen toegewezen gekregen (buffer genaamd), dat wordt gebruikt om het blok dat op een bepaald ogenblik wordt verwerkt op te slaan. Gewoonlijk wordt aan ieder bestand slechts één buffer toegewezen. Indien RESERVE 2 AREAS is gespecificeerd, worden twee buffers aan het bestand toegewezen, waarbij een blok wordt gelezen in de tweede buffer terwijl het blok in de eerste buffer wordt verwerkt. In de meeste gevallen kan dit de verwerking van het programma versnellen, maar er zijn ook situaties denkbaar waarin er juist vertraging door zou optreden. Het is aan te bevelen om de handleiding van de fabrikant te raadplegen om na te gaan wat precies het effect van deze opdracht is bij de computer waarop u werkt.

Er zijn geen opgaven bij deze paragraaf. Bestudeer nu de specificatie van het voorbeeldprogramma en codeer het aan de hand van het gegeven programmastroomschema.

J3 VOORBEELDPROGRAMMA



FIGUUR 22 Voorbeeldprogramma: schema van de computergang

Invoer

De records van MUT zien er als volgt uit:

positie 1 mutatiecode, 1 alfabetisch teken
2-6 Klantnummer, 5 cijfers
7 Kredietcode, 1 alfabetisch teken
8-16 Debet saldo, 8 cijfers, met een denkbeeldige decimale punt voor de laatste twee cijfers, en voorafgegaan door een expliciet vermeld teken.
17-46 Klantnaam, 30 tekens
47-136 Adres, 3 regels van elk 30 tekens

De records van TABEL zien er als volgt uit:

positie 1 Kredietcode, 1 alfabetisch teken
2-8 Kredietlimiet, 7 cijfers.

Uitvoer

GELDMUT heeft dezelfde recordindeling als MUT.

FOUTENVERSLAG heeft een opmaak die bestaat uit een opeenvolging van groepen regels als volgt:

Regel 1

positie 2	Mutatiecode
4-8	Klantnummer
10	Kredietcode
11	Linkerhaakje
12-20	Kredietlimiet, nul onderdrukt, komma ingevoegd
21	Rechterhaakje
23-32	Debetsaldo, niet opgemaakt
34-63	Klantnaam
65-94	Adres regel 1
96-	Boodschap "INVOER FOUT"

Regel 2

2	Eventuele asterisk
4	Eventuele asterisk
10	Eventuele asterisk
23	Eventuele asterisk
34	Eventuele asterisk
65-94	Adres regel 2

Regel 3

65-94 Adres regel 3
gevolgd door een regel wit.

De eerste pagina van het FOUTENVERSLAG moet worden voorzien van een kopregel met als tekst: "CONTROLE VAN MUTATIES - FOUTENVERSLAG" gevolgd door een regel wit. Deze pagina moet in ieder geval worden gemaakt, ook als er geen fouten zijn.

Verwerking

Het TABEL-bestand moet worden gelezen en opgeslagen in het geheugen. Er behoren 1 t/m 26 records in dit bestand aanwezig te zijn. Als het bestand leeg is, of als er meer dan 26 records worden gevonden, moet het programma worden onderbroken met de boodschap "TABEL-BESTAND LEEG" of "TABEL-BESTAND ONGELDIG" in overeenstemming met de situatie.

Ieder record in MUT moet worden gecontroleerd, waarna de geldige records worden weggeschreven naar GELDMUT. Als een veld in MUT ongeldig is, is het record ongeldig en moet dat worden weggeschreven in het FOUTENVERSLAG als een groep regels. De tweede regel van de groep moet een asterisk bevatten onder ieder ongeldig veld. De controles die worden uitgevoerd zijn:

Mutatiecode moet "I", "W" of "V" zijn (hetgeen Invoegen, Wijzigen of Verwijderen betekent).

Klantnummer moet uit 5 cijfers bestaan.

Kredietcode moet gelijk zijn aan een kredietcode in het TABEL-bestand.
Debetsaldo moet helemaal uit spaties bestaan, of uit een teken gevolgd door acht cijfers.
Debetsaldo moet uit blanco posities bestaan of + nul als mutatiecode "W" is.
Klantnaam mag niet uit blanco posities bestaan.

In tegenstelling tot de regels hierboven, moeten alle velden behalve het klantnummer uit blanco posities bestaan, wanneer de mutatiecode "V" is. Wanneer deze code "W" is, kan ieder veld anders dan klantnummer uit blanco posities bestaan.

Wanneer details van een ongeldig record worden afgedrukt, moet de overeenkomstige kredietlimiet worden afgedrukt, indien de kredietcode geldig is en niet uit blanco posities bestaat. Anders blijft het veld kredietlimiet leeg in het verslag.

Nadat verslag is gedaan van het laatste ongeldige record, of bij het begin van het verslag als er geen ongeldige records zijn, moet de volgende boodschap verschijnen:

```
GELDIGE RECORDS: INVOEGINGEN ZZZ9 WIJZIGINGEN ZZZ9
VERWIJDERINGEN ZZZ9 TOTAAL GELDIGE ZZZ9
TOTAAL DEBETSALDO VAN GELDIGE -Z,ZZZ,ZZ9.99
ONGELDIGE RECORDS: INVOEGINGEN ZZZ9 WIJZIGINGEN
ZZZ9 TOTAAL ONGELDIGE ZZZZ9
```

Deze boodschap moet evenmin als een foutengroep gesplitst worden door het pagina-einde; begin een nieuwe pagina, indien noodzakelijk, om dit opsplitsen te voorkomen.

U doet er verstandig aan uw kennis van de hoofdstukken A t/m J door verdere praktische oefening te verdiepen alvorens verder te gaan met de hoofdstukken K t/m T. In Appendix B is een bloemlezing van opgaven opgenomen: de eerste vijf opgaven kunnen worden gemaakt uitsluitend en alleen op grond van hetgeen u tot nog toe heeft geleerd.

J4 PROGRAMMA'S VAN FOUTEN ONTDOEN

Het zou bijzonder opmerkenswaardig zijn als één van de programma's, die u schrijft om u te oefenen, direct zou functioneren. U zult dus worden geconfronteerd met de taak programma's van fouten te ontdoen, d.w.z. programmafouten vast te stellen en ze vervolgens te corrigeren. De volgende opmerkingen (gebaseerd op onderzoek van J. Chan) kunnen u helpen bij een systematische vaststelling van fouten in de verwerking.

1. Bestudeer de resultatenuitlijsting (indien aanwezig). Zorg voor een duidelijk inzicht m.b.t. het symptoom, d.w.z. wat is exact het verschil tussen wat u heeft en wat u verwachtte.
2. Bestudeer de boodschappen van het operating systeem voor mogelijke aanwijzingen, bijvoorbeeld tussentijdse onderbreking, waarschuwingen, overmatige verwerkings- of invoer/uitvoertijd.
3. Controleer invoergegevensbestand(en) op formaat, blokkingsfactor en inhoud om erachter te komen of ze met het gedeclareerde formaat in het programma overeenstemmen en de inhoud hebben die u bedoelde.
4. Stel een lijst met hypotheses op wat fout aan het programma kan zijn. Een hypothese is iedere bewijsbare of weerlegbare verklaring van de fout.
5. Begin met de meest waarschijnlijke of de gemakkelijkst te toetsen hypotheses en controleer voor iedere hypothese het deel of de delen van het programma (of de gegevens of bedieningsinstructies enzovoort) die relevant zijn voor de geformuleerde hypothese.
6. Bij het toetsen van hypotheses kunnen de volgende methoden worden gehanteerd.

   a) Voorwaartse methode. Traceer op basis van de hypothese stap voor stap en voorwaarts een specifieke routine of sprong waar de fout zou kunnen zitten. Traceren betekent hier dat u doet alsof u de computer bent en alle instructies uitvoert.

   b) Achterwaartse methode. Traceer op basis van de hypothese stap voor stap en achterwaarts de instructies in de specifieke subroutine of programmeersprong; waarbij u begint op de plaats waarvan u denkt dat de fout optreedt.
7. Wanneer een fout is gelokaliseerd, controleer dan of die het symptoom afdoende verklaart. Denk eraan dat u ergens anders zult moeten zoeken indien de fout zich niet bevindt op de plek waar u hem in eerste instantie verwacht.

Schrijf bij het traceren de paragraafnamen op in de volgorde waarin u ze uitvoert evenals de waarden van variabelen die van belang zijn. Wanneer een paragraaf d.m.v. de PERFORM-instructie wordt afgehandeld, laat dan de paragraafnamen van de subroutine inspringen, totdat de subroutine is afgesloten. Ter illustratie moge het volgende ingewikkelde voorbeeld dienen (schrijf dergelijke programma's liever niet!):

```
P1. MOVE ZERO TO TELLER.
    PERFORM P2 THRU P5.
P2. PERFORM P3.
P3. ADD 1 TO TELLER.
    IF TELLER = 5 MOVE ZERO TO TELLER.
P4. ADD 3 TO TELLER.
P5. DISPLAY TELLER.
P6. ... wat is de waarde van teller?
```

Het traceren gaat als volgt:

```
P1  TELLER = 0
    P2
        P3  TELLER = 1
    P3  TELLER = 2
    P4  TELLER = 5
    P5
P2
    P3  TELLER = 6
P3  TELLER = 7
P4  TELLER = 10
P5
P6  TELLER = 10
```

ANTWOORDEN HOOFDSTUK J

Paragraaf J1

1.

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. ISITOK.
*AUTHOR. UWNAAM.
*INSTALLATION. COMPUTER SERVICE CENTRALE.
*DATE-WRITTEN. DATUM VANDAAG.
*SAMENVATTING.
*INVOER    -EEN BESTAND MET RECORDS VAN 80 POSITIES
*UITVOER   -UITLIJSTING VAN GERANGSCHIKT BESTAND
*VERWERKING-DE RECORDS WORDEN INGEVOERD, GERANGSCHIKT
*           NAAR KOLOM 1 T/M 6 (OPKLIMMEND) EN
*           AFGEDRUKT.
```

DEEL 2

Programmerings-technieken

K CONSTRUCTIES VOOR GESTRUCTUREERD PROGRAMMEREN

K1 HET SCHRIJVEN VAN GROTE PROGRAMMA'S

In Deel 1 van dit boek heb ik de aandacht gericht op de taalregels in COBOL en hoe die worden toegepast bij het schrijven van kleine programma's. Dit en het volgende hoofdstuk hebben niets te maken met 'regels' in COBOL, maar hebben betrekking op aanbevelenswaardige gewoontes die de programmeur in staat stellen op een goede manier grote programma's te schrijven.

Indien de student dit niet al heeft ontdekt, dan zal hij spoedig inzien dat het leren en toepassen van de COBOL-regels het gemakkelijkste onderdeel van programmeren is. Het verschil tussen goede en slechte programmeurs heeft weinig te maken met hun kennis van de finesses van de taal, maar veel meer met het antwoord op de vraag hoe zij hun programma's ontwerpen, documenteren en testen en met de mate waarin zij de hun gegeven opdrachten begrijpen.

Dit onderscheid is vaak zonder veel betekenis voor de student. Dat komt doordat de programmeringsproblemen, waarvoor hij zich in de opgaven ziet geplaatst, in het algemeen overzichtelijk en kort zijn: misschien maar 30 of 40 COBOL-instructies en zelden meer dan 100. De problemen bij het ontwerpen, documenteren en testen van dergelijke programma's zijn tamelijk gering. De uitlijsting van het gehele programma zal waarschijnlijk passen op twee pagina's computerpapier; de in het programma gebruikte paragrafen en gegevens kunnen gemakkelijk worden gelokaliseerd en er zijn waarschijnlijk maar een paar verschillende programmawegen die getest moeten worden.

In tegenstelling hiermee zijn COBOL-programma's, geschreven voor toepassing in de praktijk dikwijls tamelijk groot. Een programma met een gemiddelde omvang kan honderden instructies bevatten en een groot programma wel duizenden instructies. Bij programma's met een dergelijke omvang zijn de problemen m.b.t. het ontwerpen, documenteren en testen van een andere orde van grootheid. Een belangrijke methode om deze problemen aan te

pakken is dan ook, zoals we straks zullen zien, het opsplitsen van het programma in kleinere eenheden met een hanteerbare omvang.

Deze kleinere eenheden kunnen zelf programma's zijn die afzonderlijk worden gecompileerd en getest en vervolgens aan elkaar gekoppeld, zoals wordt beschreven in hoofdstuk U. Of ze kunnen onderdelen zijn van één enkele compilering. De laatstgenoemde vorm zal voorlopig worden aangehouden, maar voor mensen met ervaring die beschikken over goede mogelijkheden voor het koppelen van programma's kan de eerstgenoemde techniek de voorkeur hebben.

De in dit Deel voorgestelde methoden zijn geenszins overal aanvaard of overeengekomen. Vele commerciële programmeringsafdelingen ontwikkelen hun eigen methoden en interne regels (programmeringsstandaards). Wel representeren de in dit boek voorgestelde methoden en standaards een hanteerbare en praktische methode voor het schrijven van grote programma's die de student gemakkelijk kan aanpassen aan andere standaards die hij eventueel tegenkomt.

Hoofdstuk L (het volgende hoofdstuk) is waarschijnlijk het moeilijkste hoofdstuk van dit boek. Maar ook dit hoofdstuk, dat een inleiding is op hoofdstuk L, is niet gemakkelijk. Hopelijk bevordert de ordening van de stof uw begrip van de diverse onderwerpen, maar u moet wel beseffen dat het vele programmeurs jaren heeft gekost om ze volledig te doorgronden. U moet dus niet verwachten dat u deze hoofdstukken zult beheersen zonder aanzienlijke inspanning. Als u zich de moeite getroost die deze hoofdstukken vereisen, weet ik zeker dat u daarvan uiteindelijk geen spijt zult hebben.

De in dit hoofdstuk gebruikte voorbeelden zijn afkomstig uit paragraaf J3 en Appendix B (het eerste vraagstuk over het probleem van de bevolkingsexplosie). Het scheelt enorm als u deze voorbeelden als onderdeel van uw studie m.b.t. Deel 1 hebt uitgevoerd; indien dat niet het geval is, raad ik u aan de specificaties te bestuderen en uw oplossing te overdenken alvorens verder te gaan.

Opgave

1. Plaats voor de automatiseringsafdeling van een commercieel bedrijf de volgende programmeringsdoelen in volgorde van belangrijkheid.

   a) Minimaliseren van verwerkingstijd en geheugenbehoefte.
   b) Minimaliseren van de gebruikte hoeveelheid pennen en coderingspapier.
   c) Verkrijgen van correcte resultaten.
   d) Minimaliseren van alle moeite besteed aan het ontwikkelen en testen van een werkend programma.
   e) Minimaliseren van de totale tijd besteed aan de codering van het programma.

f) Voorzieningen aanbrengen die iemand anders in staat stellen uw programma te begrijpen, zodat hij:
   1. u kan helpen met een probleem m.b.t. de logica;
   2. uw plaats kan innemen, indien u het bedrijf verlaat of afwezig bent;
   3. het programma kan wijzigen, nadat het de produktieve status heeft verkregen als een verborgen fout aan het licht komt of om het programma aan nieuwe eisen te te laten voldoen.

K2 ONTWERPEN OM TE TESTEN

Aangezien het belangrijkste doel bij het programmeren het verkrijgen van juiste resultaten is, moet het programma zo worden ontworpen dat het testen en het bewijzen van de correctheid ervan zonder al te veel moeite kunnen worden voltooid. Nu wil het geval dat het in het algemeen ook minder tijd vergt om programma's die worden ontworpen met het testen ervan in het achterhoofd tot een produktieve fase te ontwikkelen en ook zijn de programma's veel gemakkelijker te onderhouden. Daarmee wordt dan tegelijkertijd aan twee andere belangrijke programmeerdoelen voldaan.

Een opmerkelijke gemeenschappelijke trek van veel programmeurs is hun optimisme. Zij schrijven hun programma's alsof die bij de eerste keer volledig correct zullen zijn. In de praktijk gebeurt het heel zelden dat een programma van welke omvang dan ook bij de eerste poging correct wordt opgeschreven. Ik heb meegemaakt dat zelfs ervaren programmeurs een groot programma coderen in enkele dagen en vervolgens weken of zelfs maanden besteden aan pogingen alle logische fouten te verbeteren. Als zij iets meer tijd hadden besteed aan het ontwerpen van het programma, dan zouden hun programma's een dag of wat later zijn gecompileerd. Maar deze vertraging zou vele malen zijn gecompenseerd door de vermindering van tijd, benodigd voor het opsporen van fouten en het testen.

Ik haal dit aan om u te stimuleren een defensieve instelling ten aanzien van het programmeren aan te nemen. Met een dergelijke instelling worden programma's geschreven in de verwachting (die met aan zekerheid grenzende waarschijnlijkheid blijkt uit te komen!) dat ze het niet bij de eerste keer zullen doen, en dat veel tijd en inspanning nodig zullen zijn voor het opsporen van fouten.

Een voorbeeld van hoe een dergelijke instelling kan helpen, is het volgende. Waarschijnlijk de meest voorkomende fout bij het programmeren, en zeker een van de moeilijkste om te vinden, komt voor wanneer tussenresultaten in het programma worden gebruikt of verwerkt die niet juist zijn of op onvoorziene wijze zijn vervalst. Zulke fouten zijn moeilijk te ontdekken omdat de gegevens 'onzicht-

baar' zijn wanneer we alleen de uitlijsting en de verkeerde resultaten voor ons hebben liggen.

Wat in het algemeen gebeurt is dat bij bepaalde sleutelpunten in een programma, een veronderstelling wordt gemaakt over de waarde van een tussenresultaat, bijvoorbeeld een identificatieteken, een totaal of een index. Deze beslissende veronderstellingen worden in het algemeen herkend door de programmeur en dikwijls tot onderwerp van een commentaar gemaakt, bijvoorbeeld:

```
* DE INDEX WIJST NU NAAR HET LAATSTE TEKEN VAN
* KLANTEN-NAAM
```

of

```
* HET IDENTIFICATIETEKEN WORDT ALTIJD OP NUL GEZET BIJ
* DE EERSTE UITVOERING VAN DEZE ROUTINE, enzovoort...
```

Dit is zoals het hoort. Maar toch is, zoals de zaken er nu voor staan, de enige manier om deze veronderstellingen te testen het in gedachten doorlopen van alle uitgewerkte programmalogica. Dit kan een veeleisende opgave zijn, als we zonder vergissing willen werken, in het bijzonder indien men door zijn eigen commentaar wordt misleid. Door nu DISPLAY-opdrachten (of opdrachten om fouten op te sporen - zie paragraaf Q2) in het programma op te nemen bij deze belangrijke punten kunnen we de veronderstellingen over de waarden van tussenresultaten testen. Dit is wat ik bedoel met een defensieve instelling - het ongunstigste verwachten. Na een uitvoeringstest kan de geldigheid van de veronderstellingen worden vastgesteld door de afgebeelde variabelen te onderzoeken; als alles klopt, kunnen de DISPLAY-opdrachten worden verwijderd (of de opdrachten om fouten op te sporen).

Opgave

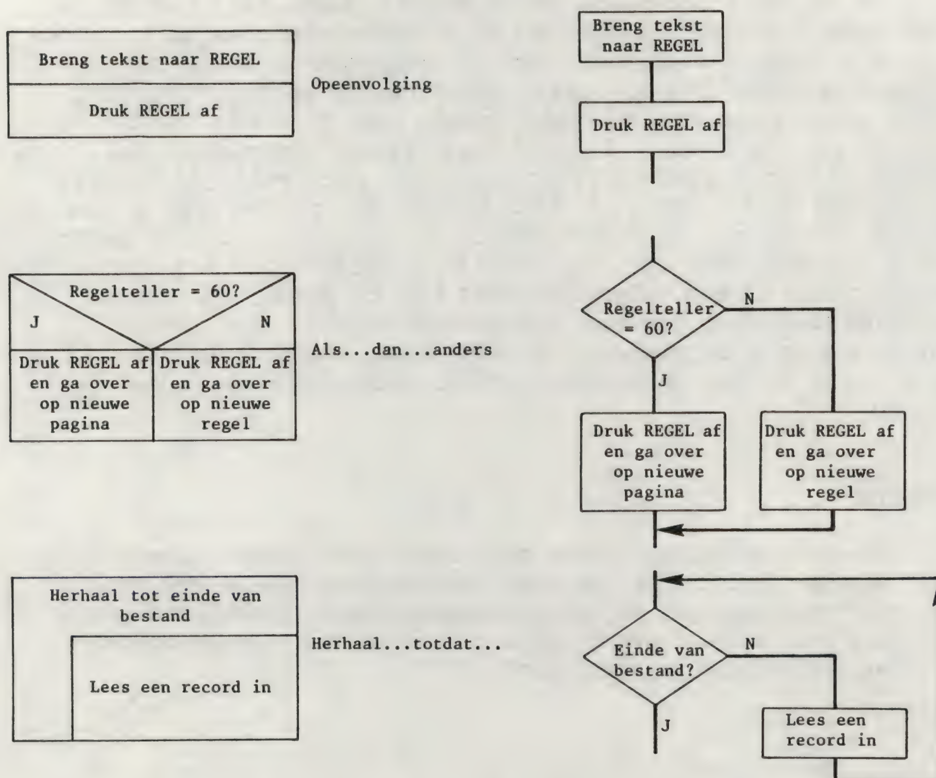
1. Wanneer DISPLAY wordt gebruikt op de zojuist voorgestelde manier, zou u dan het bedieningspaneel van de operator of de regeldrukker als afdrukeenheid willen gebruiken? Welke stappen kunt u nemen om een lawine van afdrukresultaten bij het testen te vermijden?

K3 BESTURINGSSTRUCTUREN

Het probleem van het schrijven van de Procedure Division in een COBOL-programma bestaat in het beschrijven van de fundamentele bewerkingen die moeten worden uitgevoerd (ADD, MOVE enzovoort)

en de vertakkingen en herhalingen die ertoe zullen leiden dat deze bewerkingen in de gewenste volgorde worden uitgevoerd. Wanneer u eenmaal de syntactische en semantische aspecten van de programmeertaal beheerst, is het moeilijke van programmeren hoofdzakelijk gelegen in het gebruik van de vertakkings- en herhalingsbesturing m.b.t. de fundamentele bewerkingen. De grondgedachte achter het gestructureerde programmeren is het beperken van de besturingsstructuren tot drie standaardtypes, waarvan men veronderstelt dat ze minder vatbaar voor fouten zijn dan andere. Deze drie structuren worden in figuur 24 afgebeeld.

Eigenlijk zou het nog beter zijn geheel van besturingsstructuren af te zien door het gebruik van hogere instructies of een hogere programmeertaal, waarin de programmeur slechts benoemt wat hij wil hebben en zich geen zorgen behoeft te maken over de manier waarop het resultaat wordt verkregen. STRING en



FIGUUR 24 De drie besturingsstructuren van gestructureerd programmeren. Links staan de structuurdiagrammen, rechts de gelijkwaardige programmastroomschema's

UNSTRING (Hoofdstuk O) zijn voorbeelden van hogere instructies. Andere voorbeelden worden in Deel 3 gegeven. De report writer in COBOL (Hoofdstuk S) beschikt over typische kenmerken van dit soort zogenaamde declaratieve talen. Hogere faciliteiten moeten waar mogelijk worden benut. Maar deze faciliteiten zijn niet altijd beschikbaar en zijn dikwijls niet toereikend voor het probleem.

Onder gebruikmaking van alleen de drie besturingsstructuren van figuur 24 kan ieder programma worden geschreven. Als we het probleem kunnen oplossen, terwijl we alleen deze besturingsstructuren gebruiken, en we weten verder hoe we deze structuren in COBOL moeten weergeven, dan beschikken we over een methode voor het ontwerpen van programma's. Het volgende hoofdstuk houdt zich bezig met de vraag hoe het probleem op te lossen in gestructureerde elementen. Dit hoofdstuk is gewijd aan de vraag hoe we de drie structuren in COBOL moeten weergeven. (Dit lijkt wellicht de verkeerde volgorde, maar ik denk dat u het volgende hoofdstuk beter zult begrijpen als u al tevoren precies weet, waarheen het leidt.)

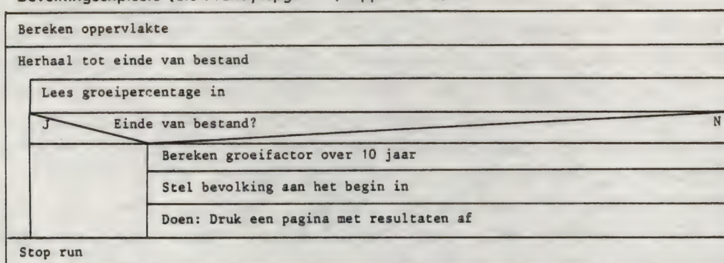
Als er van structuurschema's in tegenstelling tot stroomschema's gebruik wordt gemaakt, bestaat er geen gevaar dat de beperkte hoeveelheid besturingsstructuren die is toegestaan wordt geschonden. De structuurdiagrammen kunnen in elkaar worden ingebed zoals in figuur 25. Bekijk deze figuur goed voordat u verder gaat met de tekst.

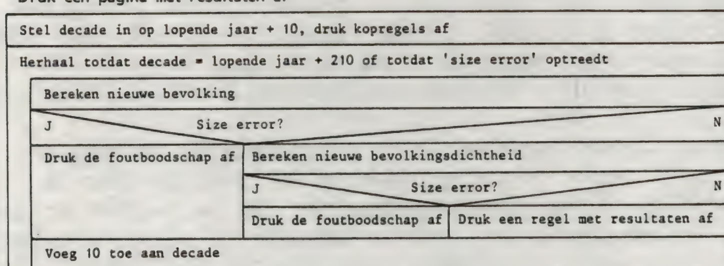
Het is een open vraag of 'bereken groeifactor over 10 jaar' en 'stel bevolking aan het begin in' thuishoren waar ze nu staan of aan het begin van 'Druk een pagina met resultaten af'. Evenzo kan 'stel decade in op lopende jaar + 10....' worden verschoven naar het hoofddiagram; trouwens, 'Druk een pagina met resultaten af' kan in zijn geheel worden opgenomen in het binnenste blok van het hoofddiagram. Al deze varianten staan op een logisch gelijkwaardig niveau; welke de voorkeur verdient, wordt aan het einde van het volgende hoofdstuk toegelicht.

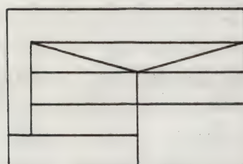
K4 OPEENVOLGING; ALS...DAN...ANDERS...

Opeenvolging

Dit is uitermate duidelijk aangezien alle gebiedende (imperatieve) instructies in COBOL achtereenvolgens worden uitgevoerd. Toch dient het volgende punt goed begrepen te worden: wanneer de fundamentele COBOL-instructies voorzien zijn van bijbehorende conditionele passages (waardoor ze conditionele en geen gebiedende instructies worden), behoort de logica van deze conditie af tot de volgende klasse van besturingsstructuren, de als...dan...anders...







Figuur 25 Dit voorbeeld laat zien hoe de drie basisstructuren in elkaar kunnen worden geschoven, of apart kunnen worden gehouden als subroutines. Ieder rechthoekig diagram moet in zijn geheel worden vervangen door een rechthoek in een ander diagram, of moet geheel apart staan; het onderste plaatje laat een voorbeeld zien dat niet is toegestaan.

De conditionele varianten van de in Deel 1 behandelde fundamentele instructies zijn:

rekenkundige instructies met de passage SIZE ERROR
READ met de passage AT END
WRITE met de passage AT END-OF-PAGE.

Als...dan...anders

Een COBOL-zin met IF...ELSE komt precies overeen met de als...dan...anders structuur. Als er niets moet worden gedaan in de 'onware' poot, wordt het ELSE-gedeelte van de zin weggelaten. Als er niets wordt gedaan in de 'ware' poot, kan de zin NEXT SENTENCE worden gebruikt; bijvoorbeeld IF REGELTELLER LESS THAN 60 NEXT SENTENCE ELSE PERFORM NIEUWE-PAGINA.

De conditionele varianten van de fundamentele instructies stemmen eveneens overeen met de als...dan structuur, maar helaas hebben ze in standaard COBOL geen 'onware' poot. ELSE in standaard COBOL past alleen bij IF. (Sommige niet gestandaardiseerde compilers paren ELSE aan iedere conditie, bijvoorbeeld READ...AT END...ELSE... . Er is nu voorgesteld dat COBOL de mogelijkheid van else-poten voor SIZE ERROR enzovoort moet hebben door een NOT-passage achter de conditie, bijvoorbeeld ADD...ON SIZE ERROR...NOT SIZE ERROR...) De huidige standaardversie levert een probleem op, wanneer er iets moet worden gedaan, zoals dikwijls voorkomt, in de 'onware' poot van een fundamentele instructie.

Een duidelijke oplossing voor simpele gevallen (d.w.z. wanneer de fundamentele instructie niet zelf ondergeschikt is aan een conditie) houdt in dat we een gegevenselement als identificatieteken instellen om het resultaat van de conditionele zin in de fundamentele instructie te signaleren. Dit identificatieteken kan vervolgens worden getest d.m.v. een IF...ELSE-zin, hetgeen op doeltreffende wijze een 'onware' poot aan de conditie toevoegt. Als voorbeeld de weergave van een deel van figuur 25 (we gebruiken W voor Waar en O voor Onwaar):

```
MOVE "O" TO INSTEL-TEKEN.
COMPUTE BEV ROUNDED = BEV * GROEI-FACTOR
ON SIZE ERROR
    MOVE "W" TO INSTEL-TEKEN.
IF INSTEL-TEKEN = "W"
    WRITE REGEL FROM FOUTMELDING AFTER 2
ELSE
    :
```

Het volgende voorbeeld is een illustratie ontleend aan het in paragraaf J3 gespecificeerde programma en laat zien hoe we een als...dan...anders, dat is ingebed in een einde-van-bestand als...dan...anders kunnen afhandelen.

```
MOVE "W" TO EVB-TEKEN
READ TABEL-BESTAND
AT END
    MOVE "W" TO EVB-TEKEN
```

```
IF EVB-TEKEN = "O"
    IF TABELINDEX LESS THAN 26
        ADD 1 TO TABELINDEX
        MOVE TABELREC TO W-S-TABELREC(TABELINDEX)
    ELSE
        DISPLAY "TABELBESTAND ONGELDIG"
        STOP RUN.
```

Als u niet helemaal gelukkig bent met dit voorbeeld, teken dan het erbij behorende structuurdiagram en overtuig uzelf van de exacte betekenis. Het ging er hier uitsluitend om hoe zo'n structuur in COBOL is weer te geven.

Wanneer de fundamentele instructie met de conditionele passage zelf verschijnt in een conditionele zin, hebben we te maken met een ingebedde conditie, waarin de ondergeschikte conditie niet over een "onware" poot beschikt. Aangezien de punt het bereik van *alle* condities markeert, kunnen we niet zo eenvoudig als hierboven eruit komen. Kijk naar het vervolg van het voorlaatste voorbeeld vanuit het structuurdiagram van figuur 25:

```
:
IF INSTEL-TEKEN = "W"
    WRITE REGEL FROM FOUTMELDING AFTER 2
ELSE
    COMPUTE DICHTHEID ROUNDED = BEV / OPPERVLAKTE
    ON SIZE ERROR
        MOVE "W" TO INSTEL-TEKEN.
IF INSTEL-TEKEN = "W"
    WRITE REGEL FROM FOUTMELDING AFTER 2
ELSE
    MOVE...(maak uitgewerkte regel klaar)
    WRITE REGEL FROM UITGEWERKTE-REGEL AFTER 2.
:
```

Het probleem is hier dat de tweede IF-zin wordt uitgevoerd zelfs indien de eerste IF-conditie waar is, in *tegenstelling* tot het structuurdiagram. De consequentie in dit voorbeeld is dat de foutboodschap tweemaal wordt afgedrukt, wanneer BEV overloopt.

Een algemene oplossing voor dit probleem is het verwijderen van ieder dubbel gebruik van identificatietekens (bijvoorbeeld door te schrijven: BEV-INSTEL-TEKEN, DICHTHEID-INSTEL-TEKEN) en het opnieuw testen van alle condities in de tweede zin, bijvoorbeeld:

```
:
IF BEV-INSTEL-TEKEN = "W"
    NEXT SENTENCE
ELSE
    IF DICHTHEID-INSTEL-TEKEN = "W"
        WRITE REGEL FROM FOUTMELDING AFTER 2
    ELSE
        MOVE...
        WRITE REGEL FROM UITGEWERKTE-REGEL AFTER 2.
:
```

Maar dit is nogal vermoeiend. De moeilijkheid is dat de *opnieuw* geteste condities altijd aanleiding geven tot een NEXT SENTENCE in de ware poot, omdat ze beogen over de gehele codering in de oorspronkelijke valse poot heen te springen. Hoewel we de codering iets kunnen vereenvoudigen doordat we de NEXT SENTENCEs kunnen verwijderen als we de condities vervangen door hun ontkenningen, is deze oplossing nog steeds tamelijk omslachtig.

Er zijn nog twee andere algemene oplossingen: (a) de voorwaartse GO TO en (b) het out-of-line plaatsen van de conditionele instructie.

In de eerstgenoemde methode moet de gemeenschappelijke activiteit die na voltooiing van welke conditionele actie dan ook moet worden uitgevoerd worden geïdentificeerd en van een label voorzien. In figuur 25 gaat het dan om de actie 'voeg 10 toe aan decade' - laten we deze VOEG-TIEN-TOE noemen. Het opnieuw testen van condities in het voorbeeld kan nu worden vermeden door een GO TO naar het label aan het einde van de geneste condities, d.w.z.:

```
    :
    MOVE "O" TO INSTEL-TEKEN.
    COMPUTE BEV ROUNDED = BEV * GROEI-FACTOR
    ON SIZE ERROR
        MOVE "W" TO INSTEL-TEKEN.
    IF INSTEL-TEKEN = "W"
        WRITE REGEL FROM FOUTMELDING AFTER 2
        GO TO VOEG-TIEN-TOE.
    COMPUTE DICHTHEID ROUNDED = BEV / OPPERVLAKTE
    ON SIZE ERROR
        MOVE "W" TO INSTEL-TEKEN.
    IF INSTEL-TEKEN = "W"
        WRITE REGEL FROM FOUTMELDING AFTER 2
        GO TO VOEG-TIEN-TOE.
    MOVE...
    WRITE REGEL FROM UITGEWERKTE-REGEL AFTER 2
VOEG-TIEN-TOE.
    ADD 10 TO DECADE.
    :
```

Hierin kan in beide IF-zinnen een ELSE de punt aan het einde vervangen, zonder dat daardoor de interne logica verandert. Maar omdat de GO TO-instructies vooruit wijzen naar de volgende paragraaf, moet alles ná de GO TO-instructie en vóór de volgende paragraaf zich bevinden in de 'onware' poot, dus is ELSE overbodig.

De voorwaarts gerichte GO TO maakt het mogelijk deze codering op te poetsen. Aangezien ELSE overbodig is geworden, behoeven we niet langer een IF-zin die ons een ELSE oplevert, te bedenken. Omdat de andere dingen gelijk blijven, kunnen we alles wat met identificatietekens heeft te maken eruit halen en het voorbeeld als volgt opnieuw coderen:

```
          :
          COMPUTE BEV ROUNDED = BEV * GROEI-FACTOR
          ON SIZE ERROR
              WRITE REGEL FROM FOUTMELDING AFTER 2
              GO TO VOEG-TIEN-TOE.
          COMPUTE DICHTHEID ROUNDED = BEV / OPPERVLAKTE
          ON SIZE ERROR
              WRITE REGEL FROM FOUTMELDING AFTER 2
          GO TO VOEG-TIEN-TOE.
          MOVE...
          WRITE REGEL FROM UITGEWERKTE-REGEL AFTER 2.
      VOEG-TIEN-TOE.
          ADD 10 TO DECADE.
          :
```

Kijken we nog eens naar het voorbeeld van figuur 25 dan zien we dat we toch een size error teken nodig hebben om te signaleren dat de herhaling in dat geval beëindigd moet worden. Het is nog steeds nodig een teken in te stellen, wanneer zich een size error voordoet; dit kan echter worden gedaan in de SIZE ERROR poten hierboven.

Indien er geen gemeenschappelijke activiteit bestaat die na voltooiing van welke conditie dan ook uitgevoerd moet worden, moet een dummy-paragraaf met de EXIT-instructie worden ingevoegd om als bestemming van de voorwaartse GO TO te dienen. Aangezien deze situatie zich alleen voordoet, wanneer de condities deel uitmaken van een blok dat wordt herhaald (of dat om een andere reden wordt uitgevoerd als een subroutine), wordt pas in de volgende paragraaf, die over herhalingen gaat een voorbeeld gegeven.

De andere algemene oplossingsmethode hield het out-of-line plaatsen van de ondergeschikte conditionele instructie in. Wanneer we teruggaan naar het oorspronkelijke voorbeeld, krijgen we de volgende oplossing:

```
    :
    MOVE "0" TO INSTEL-TEKEN.
    COMPUTE BEV ROUNDED = BEV * GROEI-FACTOR
    ON SIZE ERROR
        MOVE "W" TO INSTEL-TEKEN.
    IF INSTEL-TEKEN = "W"
        WRITE REGEL FROM FOUTMELDING AFTER 2
    ELSE
        PERFORM BEREKEN-DICHTHEID.
    ADD 10 TO DECADE.
    :
BEREKEN-DICHTHEID.
    COMPUTE DICHTHEID ROUNDED = BEV / OPPERVLAKTE
    ON SIZE ERROR
        MOVE "W" TO INSTEL-TEKEN.
    IF INSTEL-TEKEN = "W"
        WRITE REGEL FROM FOUTMELDING AFTER 2
    ELSE
        MOVE...
        WRITE REGEL FROM UITGEWERKTE-REGEL AFTER 2.
    :
```

Het gebruik van een afzonderlijk gecompileerd programma (zie Hoofdstuk U) om de functies van de subroutine uit te voeren (BEREKEN-DICHTHEID) is een bijzonder geval van deze algemene oplossing. Een afzonderlijk gecompileerd programma kan worden gebruikt daar waar de mogelijkheden optimaal zijn voor koppeling van programma's, waar de functie van de subroutine volkomen duidelijk en helder is; misschien met de kans om deze bij andere programma's opnieuw te gebruiken, en waar het zin heeft het programma als een afzonderlijke eenheid te testen. De laatste drie criteria steunen nauwelijks het opnemen van BEREKEN-DICHTHEID als een afzonderlijk programma.

Of de oplossing met GO TO of met een out-of-line subroutine de voorkeur verdient, ligt geheel aan de omstandigheden van het specifieke probleem. In het algemeen ligt de kracht van gestructureerd programmeren in de manier waarop het de spanwijdte van de aandacht, benodigd om een programma te bevatten, reduceert. De mogelijke spanwijdte van de aandacht is op haar beurt afhankelijk van de menselijke geheugencapaciteit op korte termijn en deze varieert enigszins per individu. De programmeur zou ernaar moeten streven de voor zijn programma benodigde spanwijdte van de aandacht te verkleinen, zowel voor hemzelf als voor iedere toekomstige lezer.

De spanwijdte van de aandacht wordt bij gestructureerd programmeren versmald door de mogelijkheid de blokken onafhankelijk van elkaar te bekijken. In figuur 25 bijvoorbeeld kan 'Druk een pagina met resultaten af' onafhankelijk van 'Bevolkingsexplosie' worden bekeken mits men beseft dat het groeipercentage en de bevolking in het begin op een bepaalde waarde zijn ingesteld.

Deze twee variabelen kunnen worden opgevat als parameters of onafhankelijke variabelen die het hoofdprogramma met de subroutine verbinden. Als er veel (laten we zeggen meer dan drie) van dergelijke parameters voor onze gedachten de revue moeten passeren voordat we het programma kunnen begrijpen zal de spanwijdte van de aandacht zich aanzienlijk moeten verbreden. Dit duidt op de noodzaak van het uitzien naar vereenvoudiging. Als in figuur 25 'Bereken groei-factor over 10 jaar' en 'Stel bevolking aan het begin in' worden overgebracht van het hoofdprogramma naar de subroutine, behoeft alleen het groeipercentage doorgegeven te worden als parameter. Aangezien dit het aantal parameters doet verminderen (van 2 naar 1), verdient deze oplossing de voorkeur.

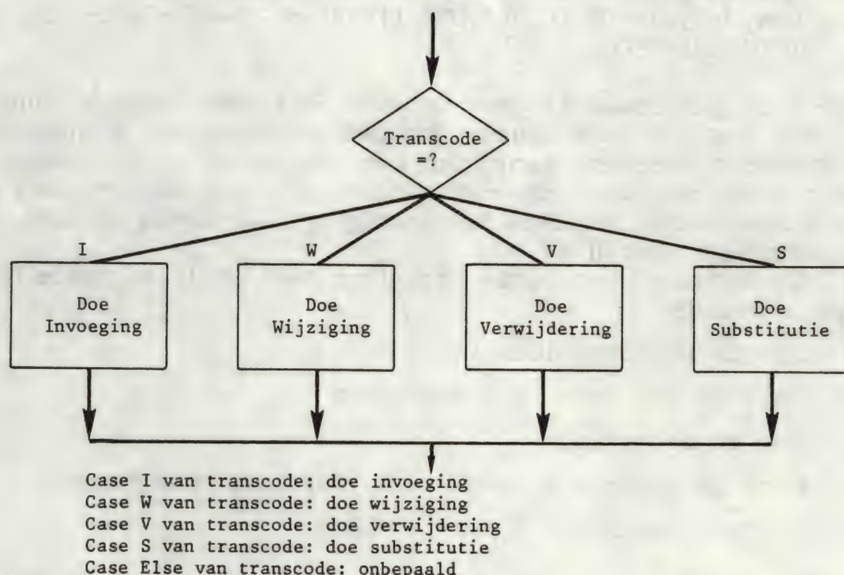
We moeten een belangrijke voorwaarde bij het principe van de onafhankelijke beschouwing noemen. Een blok kan alleen onafhankelijk worden bekeken, indien de betekenis van iedere bewerking in dat blok volkomen duidelijk is. Wanneer één van de bewerkingen bestaat in het uitvoeren van een subroutine, is de exacte betekenis daarvan gedeeltelijk afhankelijk van de gekozen woorden die zijn gebruikt om de subroutine te beschrijven: de naam van de subroutine. Dit kan door de programmeur worden geregeld. Hij dient ervoor te zorgen een subroutine op te roepen met een naam die de betekenis ervan heel precies weergeeft. Als hij geen passende exacte naam kan bedenken, kan dit een teken zijn dat de subroutine geen logisch geheel vormt.

De door de COBOL-werkwoorden aangegeven basisbewerkingen zijn volkomen duidelijk, aangezien ze in het COBOL-handboek worden beschreven. Een gelijkwaardige nauwkeurigheid wordt bij subroutine-bewerkingen zelden bereikt, zelfs niet wanneer de naam met zorg is vastgesteld. We hebben bijvoorbeeld al twee verschillende interpretaties van de subroutine 'Druk een pagina met resultaten af' gezien: de interpretatie van figuur 25 en de aangepaste interpretatie hierboven. Een perfect begrip van andermans programma wordt alleen bereikt door het te verstaan in termen van basisbewerkingen. Hoewel de auteur van het programma van figuur 25 het hoofdprogramma volkomen kan overzien, moet de lezer daarvoor ook de subroutine in zich opnemen. De noodzaak om de subroutine in zich op te nemen verbreedt de vereiste spanwijdte van de aandacht waardoor het nut van onafhankelijke beschouwing verloren gaat. Behalve in het geval van een subroutine waarvan het doel heel goed wordt begrepen (het gaat bijvoorbeeld om een heel vertrouwde of standaardroutine of de routine voert een (wiskundige) functie uit die overal herkend wordt) behoort het een programmeringsdoel te zijn om de diepte van het nesten van subroutines te beperken.

De moraal van dit verhaal is aan te tonen dat de voorwaartse GO TO-oplossing de voorkeur verdient in de geschetste omstandigheden. Maar bij andere omstandigheden, wanneer de functie van de subroutine beter wordt beschreven of wanneer de bestemming van de GO TO zeer ver verwijderd is en daardoor de aandacht zich moet rekken, is de oplossing met de subroutine beter.

Opgaven

1. Vele programmeurs gebruiken graag een vierde constructie bij hun gestructureerd programma-ontwerp, namelijk de 'case' constructie (ook wel 'selection' genaamd). In principe is dit een lijst met te nemen acties overeenkomstig de waarde van een of andere variabele; een voorbeeld wordt gegeven in figuur 26.



FIGUUR 26 Paragraaf K4, Opgave 1. De case-constructie.

Deze constructie kan worden weergegeven door een andere constructie voor gestructureerd programmeren: hoe?

2. Welke alternatieven ziet u voor het invoeren van een case-constructie in COBOL (a) wanneer de case-variabele alfanumeriek is; (b) wanneer de case-variabele een geheel getal is in de reeks 1 t/m n, waarin n het aantal cases voorstelt? Wat is het beste alternatief?

K5 REPEAT...UNTIL

Deze constructie zal in dit boek worden opgevat als exact gelijkwaardig aan de COBOL-instructie PERFORM...UNTIL (en in voorkomende omstandigheden aan zijn variant PERFORM...VARYING...FROM...BY...UNTIL). De herhalings-

blokken in figuur 25 kunnen achtereenvolgens worden uitgedrukt als volgt:

```
PERFORM READ-GROEI-ETC UNTIL EVB-TEKEN = "W"
```

en

```
PERFORM BER-BEV-EN-DRUK-AF VARYING DECADE FROM
lopende-jaar+10 BY 10 UNTIL DECADE = lopende-jaar+210
OR INSTEL-TEKEN = "W".
```

Hoewel de gelijkheidstest met 'lopende-jaar+210' duidelijk zinnig is, zou een test met 'niet minder dan 200' een kleinere spanwijdte van de aandacht vereisen, aangezien naar gegevens in het probleem wordt verwezen. Maar de 'niet minder dan'-stijl behoort niet te worden gebruikt, als deze het gevolg is van slordig denken m.b.t. de beëindigingsconditie.

De logica achter iedere herhaling moet in de volgende fasen worden bedacht:

1. Stel de begincondities vast
2. Maak de test voor de beëindiging
3. Doe de bewerking
4. Werk de controlevariabelen bij voor zover aangegeven
5. Herhaal vanaf fase 2.

De PERFORM...VARYING... voorziet automatisch in een deel van deze logica en wordt daarom aanbevolen. We kunnen er echter niet altijd alle beginwaarden mee instellen noch alle vereiste controlevariabelen mee bijwerken. De twee bovengenoemde voorbeelden moeten daarom worden voorafgegaan door respectievelijk:

```
MOVE "O" TO EVB-TEKEN
```

en

```
MOVE "O" TO INSTEL-TEKEN
```

Onder verwijzing naar figuur 25 zal het duidelijk zijn dat, wanneer het bestandseinde is bereikt, er niets moet worden gedaan in het herhalingsblok. Dus:

```
READ-GROEI-ETC.
    READ GROEIBESTAND
    AT END
        MOVE "W" TO EVB-TEKEN.
    IF EVB-TEKEN = "O"
        COMPUTE GROEI-FACTOR ROUNDED = ....
        MOVE 4000000000 TO BEV
        PERFORM DRUK-EEN-PAGINA-MET-RESULTATEN-AF.
```

Hier doet zich een complicatie voor, als een size error-passage wordt toegevoegd aan de COMPUTE-instructie, zoals werd aanbevolen in paragraaf C3. Natuurlijk kan deze complicatie worden opgeheven als de betekenis van 'Druk een pagina met resultaten af' wordt verbreed, zoals in de laatste paragraaf werd toegelicht - maar laten we ons ter wille van het voorbeeld aan de bovenstaande versie houden.

Er bestaat geen instructie om van een label te voorzien t.b.v. een voorwaartse GO TO, dus er moet een EXIT-paragraaf worden toegevoegd, zoals in paragraaf D5 werd besproken. Dit dwingt tot het gebruik van de PERFORM...THRU variant; om dit te vermijden kunnen we van een herhalingsblok een SECTION maken. Wanneer een SECTION met PERFORM wordt uitgevoerd, worden alle paragrafen tot en met de laatste in de section uitgevoerd; de besturing keert terug na uitvoering van de laatste paragraaf, evenals dat bij een PERFORM...THRU die de eerste en laatste paragraaf ervan zou hebben genoemd, het geval zou zijn geweest.

Het binnenste blok van ons programma kan nu worden voltooid:

```
READ-GROEI-ETC SECTION.
RG.
    READ GROEIBESTAND
    AT END
        MOVE "W" TO EVB-TEKEN
        GO TO RG-EXIT.
    COMPUTE GROEI-FACTOR ROUNDED = ....
    ON SIZE ERROR
        DISPLAY "PROGRAMMAFOUT IN RG - PROGRAMMA AFGEBROKEN"
        STOP RUN.
    MOVE 4000000000 TO BEV.
    PERFORM DRUK-EEN-PAGINA-MET-RESULTATEN-AF.
RG-EXIT.
    EXIT.
```

Het gebruik, zoals hier, van STOP RUN in een out-of-line paragraaf moet worden beperkt tot als 'abnormaal' beschouwde eindes.

In de praktijk is het zo dat een bijzondere actie nodig is als een invoerbestand leeg is. Dat heeft tot gevolg dat de eerste READ-instructie van de andere wordt onderscheiden. (D.w.z. voer de bijzondere actie uit, indien einde van bestand wordt ontdekt bij de eerste READ. In het andere geval: verwerk het record. Wanneer einde van bestand wordt ontdekt bij een van de volgende READ's voer dan de normale actie bij einde van bestand uit.)

Deze situatie wordt behandeld door de eerste READ-instructie in-line te zetten en opvolgende READ-instructies in een herhalingsblok te plaatsen. Aangezien er reeds een record is ingelezen bij de ingang naar de subroutine, moet de logica van de subroutine eerst het zojuist ingevoerde record verwerken en daarna het

volgende record lezen. Een gelukkige omstandigheid is dat hierdoor de voorwaartse GO TO wordt geëlimineerd die verbonden is met het bestandseinde in de subroutine (en daardoor de behoefte aan een EXIT-paragraaf en daardoor weer de rechtvaardiging voor een SECTION). Bekijk het voorbeeld van de bevolkingsexplosie bijvoorbeeld eens met de nieuwe eis dat een boodschap moet worden gezonden aan de operator, als het invoerbestand leeg is. (Dit voorbeeld is ook een illustratie van de aangepaste versie van 'Druk een pagina met resultaten af'.)

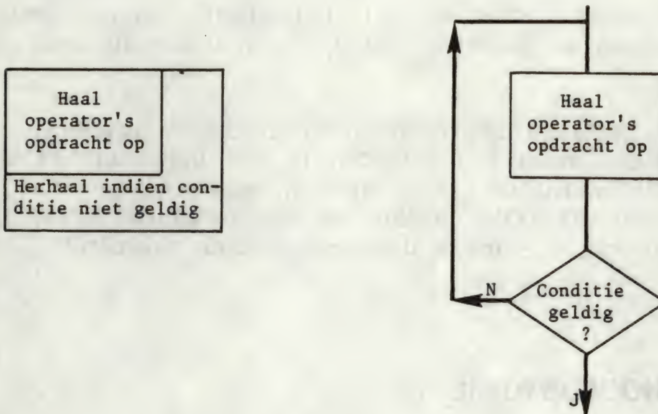
```
PROCEDURE DIVISION.
HOOFD SECTION.
MI.
    OPEN INPUT GROEIBESTAND.
    MOVE "O" TO EVB-TEKEN.
    READ GROEIBESTAND
    AT END
        MOVE "W" TO EVB-TEKEN.
    IF EVB-TEKEN = "W"
        DISPLAY "LEEG GROEIBESTAND - PROGRAMMA AFGEBROKEN"
    ELSE
        OPEN OUTPUT AFDRUKBESTAND.
        PERFORM VERWERK-GROEIPERC UNTIL EVB-TEKEN = "W"
        CLOSE AFDRUKBESTAND.
    CLOSE GROEIBESTAND.
    STOP RUN.
VERWERK-GROEIPERC.
    PERFORM DRUK-EEN-PAGINA-MET-RESULTATEN-AF.
    READ GROEIBESTAND
    AT END
        MOVE "W" TO EVB-TEKEN.
DRUK-EEN-PAGINA-MET-RESULTATEN-AF.
    COMPUTE GROEIFACTOR ROUNDED = ....
    ON SIZE ERROR
        DISPLAY ....
        STOP RUN.
    MOVE 4000000000 TO BEV.
    : } afdrukken
    : } kopregels
    MOVE "O" TO INSTEL-TEKEN.
    PERFORM BER-BEV-EN-DRUK-AF VARYING DECADE FROM ....
    BY 10 UNTIL DECADE = .... OR INSTEL-TEKEN = "W".
BER-BEV-EN-DRUK-AF SECTION.
    :
```

De plaats waar de beginwaarden aan GROEI-FAKTOR en BEV worden toegekend stemmen overeen met de bespreking tot nog toe. De enige wijziging die ik nu nog zou willen maken is de verplaatsing van deze opdrachten tot ná de afdruk van de kopregels om zo overeenstemming te bereiken met de goede algemene regel, dat

als parameters gebruikte variabelen zo dicht mogelijk bij het punt waar de subroutine wordt aangeroepen, op een bepaalde waarde worden ingesteld. Deze wijziging verandert de logica enigszins wanneer een size error optreedt. Als de size error optreedt resulteert dit in het afbreken van het programma en of in dat geval de kopregels wel of niet afgedrukt moeten worden is niet duidelijk.

Opgaven

1. Figuur 27 is een illustratie van een andere constructie die nogal eens voor gestructureerd programmeren gepropageerd wordt: de doe-eenmaal-en-herhaal...totdat... . Geef twee manieren aan waarop dit kan worden bereikt onder gebruikmaking van de basisconstructies.



FIGUUR 27 Paragraaf K5, Opgave 1

(Bij gestructureerd programmeren wordt frequent gewerkt met do...while of while...do i.p.v. PERFORM...UNTIL. Dit is minder goed in COBOL weer te geven, aangezien voor weergave van de while-instructie met behulp van de PERFORM...UNTIL van COBOL de ontkenning van de conditie uit de while in de PERFORM-instructie gebruikt moet worden, hetgeen weer de spanwijdte van de aandacht verbreedt. Er zijn schrijvers die ervan uitgaan dat repeat...until dezelfde betekenis heeft als de constructie van figuur 27; ook wordt deze constructie wel eens opgevat als een do...until of een do-while-do.

Deze situatie is ronduit ongelukkig. Welk werkwoord (repeat, do, enzovoort) wordt gebruikt, is niet echt belangrijk. Een verstandige regel zou de volgende zijn: als de conditie aan het begin wordt genoemd (until conditie do...,

while not conditie repeat...), is de test bedoeld om als eerste te worden uitgevoerd; en als de conditie aan het einde wordt genoemd, is de test bedoeld om als laatste te worden uitgevoerd. Maar deze gedachte is niet overal doorgedrongen en daarom zal in dit boek iedere vermelding van 'until' inhouden: 'test eerst' op de manier van COBOL.)

2. Sommige kleine compilers beschikken niet over de UNTIL-versie van de PERFORM-instructie. Op de meeste computers gebruikt PERFORM heel wat meer computertijd dan GO TO en in zeldzame gevallen waarbij een programma een zeer dichte lus heeft die vele duizenden keren moet worden uitgevoerd kan het wenselijk zijn enige aanpassing te verrichten en GO TO te gebruiken i.p.v. PERFORM. Hoe zou u in het algemeen een programma coderen, indien u GO TO moest gebruiken i.p.v. PERFORM?

3. Schrijf de codering die het TABEL-bestand van paragraaf J3 leest en opslaat en hanteer hierbij de in dit hoofdstuk gegeven richtlijnen.

4. Soms wordt gestructureerd programmeren gelijkgesteld met programmeren zonder GO TO. Dat is niet helemaal het standpunt van dit hoofdstuk, maar er zijn beperkingen m.b.t. reikwijdte van GO TO's, indien de hier behandelde richtlijnen worden opgevolgd. Kunt u die beperkingen noemen?

ANTWOORDEN HOOFDSTUK K

Paragraaf K1

1. De meeste mensen stemmen er gemakkelijk mee in dat (c) het belangrijkste is en (b) het minst belangrijk en dat (d) belangrijker is dan (e). Mijn argument ten gunste van doeleinde (f) is dat, als de daarin opgesomde zaken niet kunnen worden gedaan, de doeleinden (c) en (d) de neiging zullen hebben onbereikbaar te worden. Daarom zet ik (f) op de tweede plaats.

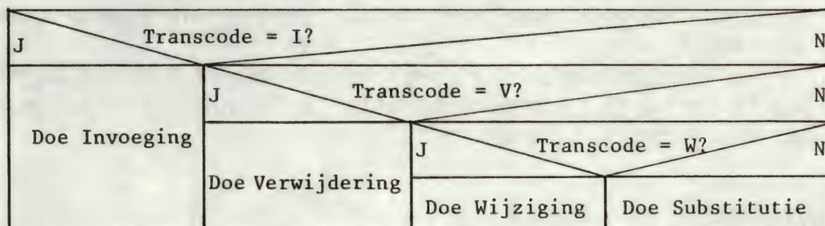
   Resteert nog een bespreking van de relatieve waarde van (a) en (d). Het is zeker dat de kosten van het ontwikkelen van een programma hoog genoeg zijn om (d) belangrijker te vinden dan (a), als (a) niet zeer dringend gewenst is. Onder dat voorbehoud is mijn lijstje: (c) (f) (d) (a) (e) (b).

Paragraaf K2

1. De regeldrukker. Als de apparatuur voor het afbeelden wordt gevormd door de beeldschermterminal van de programmeur, is bij de meeste installaties de kans groter dat het afgebeelde niet zo grondig wordt gecontroleerd als gebeurd zou zijn bij het verschijnen op harde kopieën. Het bedieningspaneel van de operator wordt alleen gebruikt voor zeer kleine gegevenshoeveelheden, en dan nog alleen voor gegevens waarvoor van de operator wordt verwacht dat hij er iets mee doet of ze vastlegt. Als de af te beelden variabele in een lus zit, kan dat uitmonden in een zeer grote hoeveelheid afdrukresultaten; u kunt de operator verzoeken het programma voortijdig na bijvoorbeeld x bladzijden of seconden te beëindigen. U kunt ook de DISPLAY-instructie conditioneel maken m.b.v. een teller.

Paragraaf K4

1. Zie figuur 28. Dit is duidelijk niet zo bevredigend, wanneer het aantal gevallen groot is.



FIGUUR 28 Antwoord bij Opgave 1, paragraaf K4

2. a) (i)

```
IF TRANSCODE = "I"
    PERFORM INVOEGING
ELSE
    IF TRANSCODE = "V"
        PERFORM VERWIJDERING
    ELSE
        IF TRANSCODE = "W"
            PERFORM WIJZIGING
        ELSE
            PERFORM SUBSTITUTIE.
```

```
(ii) IF TRANSCODE = "I"
         PERFORM INVOEGING.
     IF TRANSCODE = "V"
         PERFORM VERWIJDERING.
     IF TRANSCODE = "W"
         PERFORM WIJZIGING.
     IF TRANSCODE = "S"
         PERFORM SUBSTITUTIE.
```

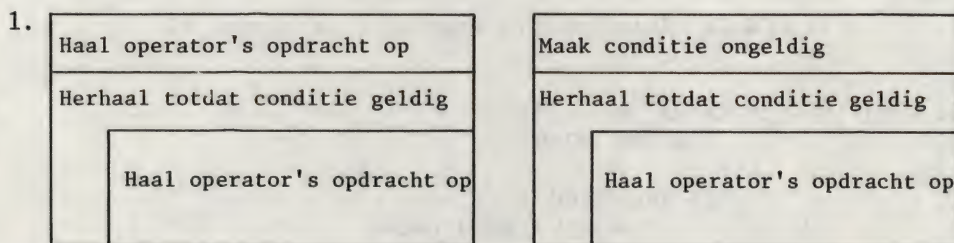
```
b)       GO TO 1 2 N DEPENDING ON CASE.
     1.  PERFORM PROCES-1.
         GO TO VERVOLG.
     2.  PERFORM PROCES-2.
         GO TO VERVOLG.
     N.  PERFORM PROCES-N.
     VERVOLG.
```

Antwoord (a)(ii) is heel eenvoudig en aantrekkelijk wanneer efficiëntie geen overweging is. De efficiëntie van (a)(ii) kan worden vergroot door aan de eerste 3 performs een voorwaartse GO TO toe te voegen naar een vervolg-paragraaf en door de performs te rangschikken in de volgorde van waarschijnlijkheid waarmee ze worden uitgevoerd. Als de andere methoden worden gebruikt, is een verklarend commentaar wenselijk.

Er bestaat natuurlijk een subtiel verschil in de logica van (a)(i) en (a)(ii) hierboven (welk?); ik ben ervan uitgegaan dat dit verschil niet belangrijk is.

Paragraaf K5

1.



FIGUUR 29 Antwoord bij Opgave 1, paragraaf K5

2. Ik zou bij mijn ontwerp van het programma gebruik maken van gestructureerde constructies (alsof PERFORM...UNTIL beschikbaar was). Waar ook maar een PERFORM...UNTIL zou verschijnen, zou ik een kleine lus aanbrengen volgens de 5-stappen logica met daarbinnen als bewerking de enkelvoudige PERFORM.

Indien verdere afstemming wordt gewenst, kan de enkelvoudige PERFORM worden vervangen door de instructies waar hij uit bestaat. Dit kan de introductie van een verzamelparagraaf met zich meebrengen en een andere paragraaf voor een voorwaartse GO TO om uit de herhaling te komen. Aangezien er een aanmerkelijk verlies in helderheid door deze afstemming optreedt, is de gulden regel: ga niet verder met afstemmen dan noodzakelijk is en breng vervolgens de PERFORM...UNTIL in als commentaar.

3.

```
        OPEN INPUT TABEL.
        READ TABEL
        AT END
            DISPLAY "TABELBESTAND LEEG"
            STOP RUN.
        MOVE "O" TO EB-TABEL.
        PERFORM VERWERK-TABELREC VARYING TABELSUB FROM 1 BY 1
        UNTIL TABELSUB GREATER THAN 26 OR EB-TABEL = "W".
        CLOSE TABEL.
        IF EB-TABEL NOT = "W"
            DISPLAY "TABELBESTAND ONGELDIG"
            STOP RUN.
        :
    VERWERK-TABELREC.
        MOVE TABELREC TO W-S-TABELREC(TABELSUB).
        READ TABEL
        AT END
            MOVE "W" TO EB-TABEL.
```

4. Voorwaartse GO TO-instructies zijn altijd gericht op een gemeenschappelijk verzamelpunt. U zult nooit voorwaartse GO TO-instructies aantreffen die haasje-over spelen. Een achterwaartse GO TO, indien die al gebruikt wordt, zal zich geheel binnen of geheel buiten het bereik van iedere andere achterwaartse GO TO (of blok met een perform-instructie) bevinden.

L GESTRUCTUREERD ONTWERPEN VAN PROGRAMMA'S

L1 DE HERKENNING VAN GEGEVENSSTRUCTUREN

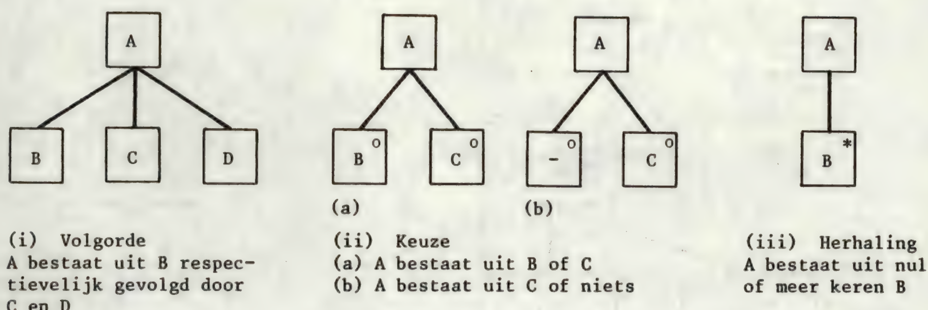
Het vorige hoofdstuk ging over het vertalen van constructies voor gestructureerd programmeren in COBOL-codering. Dit hoofdstuk houdt zich bezig met het vertalen van het probleem in constructies voor gestructureerd programmeren.

M. Jackson legde in zijn boek *Principles of Program Design* (Academic Press, 1975) de nadruk op de relatie tussen gegevensstructuren en programmastructuren. Wil de programmeur een beslissing nemen over de programmastructuur, dan zal hij eerst de bij het programma betrokken gegevensstructuren moeten doorzien. De gegevensstructuren, waarnaar wordt verwezen, zijn gewoonlijk de invoer- en uitvoerbestanden, hoewel in minder vaak voorkomende gevallen een interne tabel moet worden geanalyseerd.

De in dit hoofdstuk toegelichte methode is daarom als volgt:

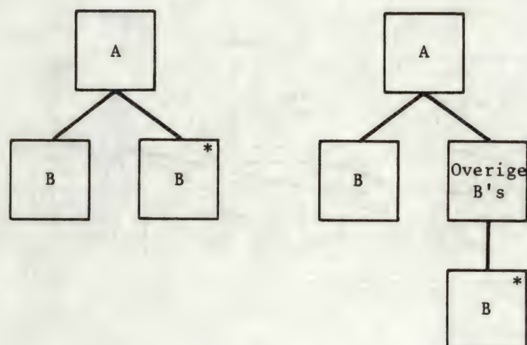
1. analyseer de gegevens in het probleem;
2. maak op basis van deze analyse een diagram van de *gegevens*structuur;
3. maak een *programma*structuur van het in hoofdstuk K beschreven type;
4. maak het COBOL-programma.

In de praktijk kunnen de gegevensstructuren dikwijls eenvoudig worden weergegeven door een hiërarchie van drie elementaire constructies: volgorde, keuze en herhaling. Jackson's schematische blokken voor deze drie structuren worden in figuur 30 afgebeeld:



FIGUUR 30 Gegevensstructuren. Als we het aantal keuzeblokken in de tweede mogelijkheid (ii) doen toenemen ontstaat een gegevensstructuur van het 'case'-type.

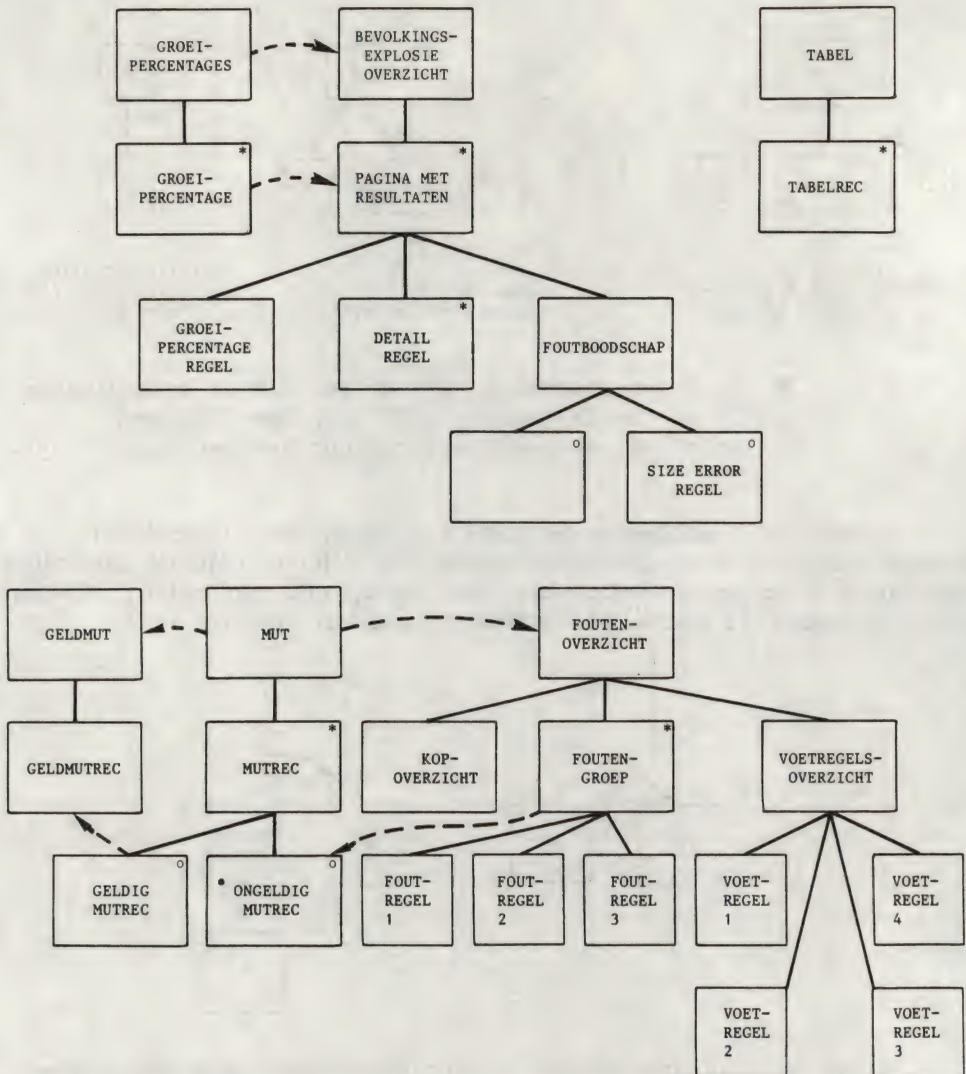
In de hier beschreven methode kan ieder blok door iedere andere constructie worden vervangen. Als bijvoorbeeld de herhaling uit één of meer keren B bestaat, kan dit op elk van beide manieren, zoals in figuur 31 wordt geïllustreerd, worden beschreven.



FIGUUR 31 Illustratie van de situatie waarin A wordt gevolgd één of meer keren B. Ieder van beide constructies is geoorloofd.

Het hoogste niveau van een gegevensstructuur is in het algemeen het bestand. De aanwezigheid van recordtypen in het bestand wordt geïllustreerd door de gegevensstructuurblokken ondergeschikt aan het bestand bij te voegen. Figuur 32 illustreert sommige situaties die we al zijn tegengekomen (de stippellijnen in deze figuur worden in de volgende paragraaf toegelicht).

Merk op dat, hoewel er slechts één type MUTREC bestaat, de geldige records anders dan de ongeldige records moeten worden verwerkt. Steeds wanneer er sprake is van een recordtype waarvan



FIGUUR 32 Gegevensstructuren bij de opgave bevolkingsexplosie (Appendix B) en het valideringsprogramma (paragraaf J3). Merk op dat TABEL weliswaar ongeldig is als het leeg is, maar dat van het valideringsprogramma verondersteld wordt deze ongeldigheid te behandelen. Het programma moet er dus nul of meer keren een TABELREC verwachten. Het FOUTENOVERZICHT moet een kop en een staartstuk hebben; het middenstuk van het overzicht kan leeg zijn. Het is niet duidelijk of er een FOUTENOVERZICHT moet worden geproduceerd als TABEL ongeldig is; waarschijnlijk is het beter het dan niet te produceren. In dat geval wordt FOUTENOVERZICHT een facultatief bestand, maar het is niet nodig dit kenmerk in het diagram vast te leggen omdat dit diagram de structuur van FOUTENOVERZICHT weergeeft als het wel wordt geproduceerd.

de records op grond van een of ander criterium op een aantal verschillende manieren behandeld moeten worden, is het, gezien het doel van het diagram voor de gegevensstructuur, aan te bevelen dit in het diagram aan te geven door daarin even zovele recordtypen in te voeren.

Opgave

1. Schets het diagram voor de gegevensstructuur ten behoeve van het probleem van de studentenaanwezigheid (Appendix B).

L2 HET AFLEIDEN VAN DE PROGRAMMASTRUCTUUR IN EEN EENVOUDIGE SITUATIE

We gaan uit van de bij het probleem betrokken gegevensstructuren m.b.t. invoer en uitvoer (zie figuur 32). Soms, zoals bij het valideringsprobleem, bestaat het programma uit een aantal fasen, in welk geval men de fasen onafhankelijk van elkaar kan beschouwen. Een fase wordt in het algemeen gekenmerkt door de mogelijkheid één of meer bestanden af te sluiten, voordat een ander bestand (of bestanden) wordt geopend. Zo kan de opslag van het TABEL-bestand onafhankelijk van de verwerking van het MUT-bestand worden bekeken.

Vervolgens tekenen we pijlen die allemaal een één-op-één verhouding aangeven tussen een invoerblok en een uitvoerblok (zie stippellijnen in figuur 32). Zo produceert één MUT-bestand één GELDMUT-bestand en één FOUTEN-OVERZICHT-bestand. Eén MUTREC produceert één GELDMUTREC en één ONGELDIG MUTREC produceert een FOUTEN-GROEP.

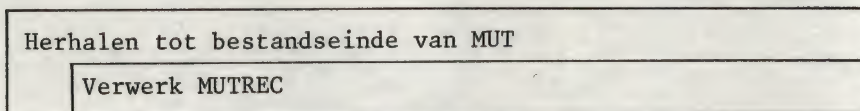
Hierna is het de bedoeling een diagram voor de *programma*-structuur (hoofdstuk K) te schetsen, dat alle blokken van het diagram voor de *gegevens*structuur verwerkt. Zodra we een blok uit het diagram voor de gegevensstructuur hebben verwerkt in het diagram voor de programmastructuur, kruisen we dit blok af. We ontwerpen het diagram voor de programmastructuur door uit te gaan van de gegevensstructuur van het invoerbestand en dit door te werken; steeds wanneer er een blok is, moeten we dat 'verwerken'. Als het een opeenvolgingsblok is met een uitgang, moeten we dat zodanig 'verwerken' dat de uitvoer wordt 'geproduceerd'. Wanneer het een herhaling is, moeten we 'repeat...until...' gebruiken. Wanneer het een keuze betreft, moeten we gebruik maken van 'if...then...else' zoals hieronder wordt geïllustreerd.

Eerste stap

Verwerk MUT om GELDMUT en FOUTEN-OVERZICHT te produceren

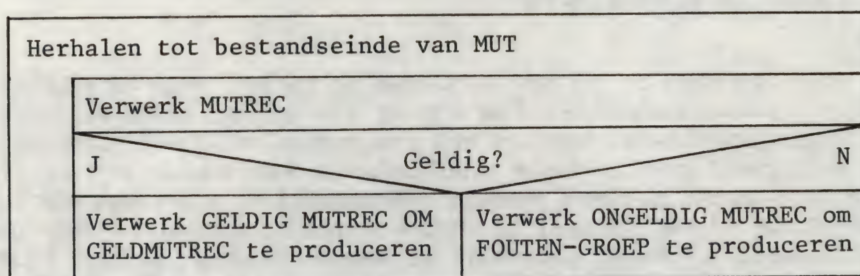
Dit is een omschrijving van de gehele programmabewerking. De resterende analyse bestaat uit een verdere uitwerking van deze bewerking. Kruis MUT, GELDMUT en FOUTEN-OVERZICHT af.

Volgende stap:



FIGUUR 33 Programmastructuur m.b.t. MUTREC

Kruis MUTREC af. De volgende stap, de keuze, is ondergeschikt aan de herhaling:



FIGUUR 34 Programmastructuur na behandeling met de keuze

Kruis GELDIG MUTREC, GELDMUTREC, ONGELDIG MUTREC en FOUTEN-GROEP af.

We hebben de invoerblokken nu allemaal gehad en moeten nog rekenschap afleggen van de niet afgekruiste uitvoerblokken. De uitvoerblokken zullen worden verwerkt op één van de volgende drie manieren:

a) als een bewerking vóór (of na) een al beschreven bewerking;

b) als een uitbreiding van een al beschreven bewerking;

c) door middel van doe...ingeval... of eventueel een andere methode.

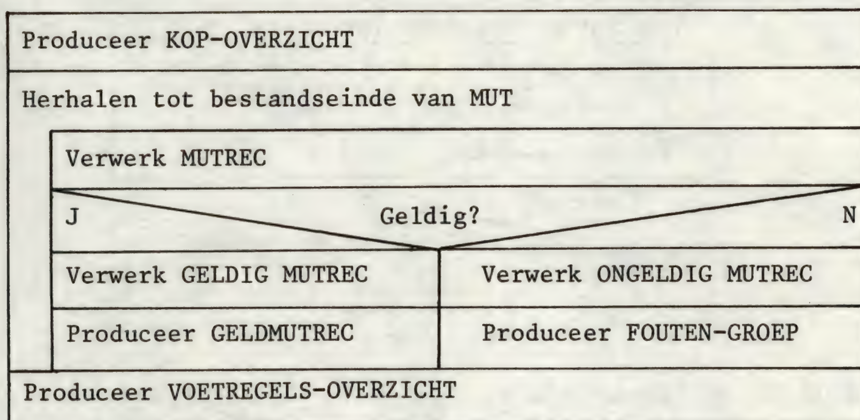
Het laatste geval wordt behandeld in de volgende paragraaf.

Als de uitvoer een keuze of herhaling is, moet die worden geproduceerd onder de besturing van de overeenkomstige gestructureerde constructie, zoals tevoren. Een uitbreiding kan òf in een afzonderlijk structuurdiagram worden geplaatst (als een subroutine) òf de oorspronkelijke bewerking vervangen.

Als we een en ander toepassen op figuur 32, is het duidelijk

dat KOP-OVERZICHT wordt geproduceerd precies *voor* het door figuur 34 beschreven proces en dat VOETREGELS-OVERZICHT precies *erna* wordt geproduceerd. Alle andere blokken betreffen uitbreidingen met opeenvolgingsopdrachten van respectievelijk FOUTEN-GROEP en VOETREGELS-OVERZICHT. Zo ontstaat dan (zie figuur 35) het volledige diagram voor de programmastructuur van het valideringsprogramma uit figuur 32:

Verwerk MUT om GELDMUT en FOUTEN-OVERZICHT te produceren



Produceer FOUTEN-GROEP

Produceer FOUTREGEL 1
Produceer FOUTREGEL 2
Produceer FOUTREGEL 3

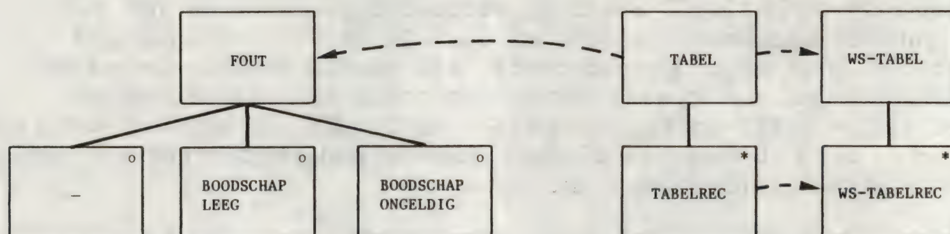
Produceer VOETREGELS-OVERZICHT

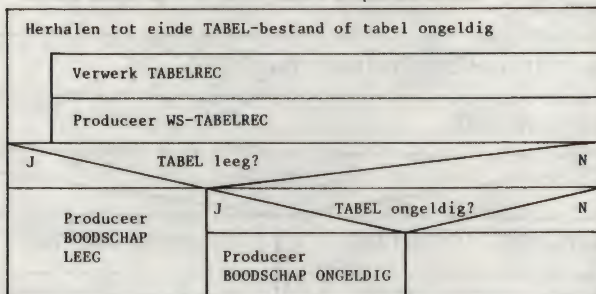
Produceer VOETREGEL 1
Produceer VOETREGEL 2
Produceer VOETREGEL 3
Produceer VOETREGEL 4

FIGUUR 35 Structuurdiagram van de tweede fase van het valideringsprogramma

Met de gegeven methode (zie het vorige hoofdstuk) van het vertalen van structuurdiagramman in codering bent u waarschijnlijk al in staat dit programma te coderen. Laten we echter ter wille van het voorbeeld doorgaan met deze methodische aanpak.

Figuur 36 laat de met de eerste fase van het programma overeenstemmende gegevensstructuren en programmastructuur zien. De working storage tabel en de eventuele foutboodschappen worden als uitvoergegevensstructuren behandeld.





FIGUUR 36 De eerste fase van het valideringsprogramma. De tweede fase wordt pas uitgevoerd, indien de tabel geldig is. Daarom hoort in het in deze figuur verkregen lege blok 'Verwerk MUT om GELDMUT en FOUTEN-OVERZICHT te produceren' te worden geplaatst. Als de tweede fase niet conditioneel was geweest had van een derde programmastructuur gebruik gemaakt kunnen worden om de fasen van het programma aan elkaar te koppelen. Deze zou slechts uit de twee opeenvolgingsbewerkingen Verwerk TABEL... en Verwerk MUT... hebben bestaan.

We gaan nu verder detailleren. Laten we aannemen dat we in working storage de volgende records hebben beschreven: KOP-OVERZICHT, FOUTREGEL-1 t/m FOUTREGEL-3, VOETREGEL-1 t/m VOETREGEL-4 en WS-TABELREC OCCURS 26 TIMES. De records TABELREC, MUTREC en GELDMUTREC hoeven alleen in de File Section te worden beschreven, omdat ze geen enkele constante bevatten waarvan de waarde door het programma moet worden bepaald. We zullen ook een File Section record nodig hebben voor FOUTEN-OVERZICHT; laten we dit REGEL noemen.

Bovendien kunnen we verwachten dat we een identificatie-element nodig hebben voor iedere conditie die wordt getest in het programmastructuurdiagram, waar de conditie niet direct verwijst naar een invoergegevenselement. Misschien kunnen een aantal identificatietekens later worden verwijderd, wanneer we de codering gaan zuiveren zoals wordt toegelicht in de volgende paragraaf. Maar laten we van de slechtst denkbare situatie uitgaan. Nadere

beschouwing van de figuren 35 en 36 levert de volgende lijst met identificatietekens op:

```
EB-MUT              PICTURE X
MUTREC-GELDIG       PICTURE X
EB-TABEL            PICTURE X
TABEL-ONGELDIG      PICTURE X
TABEL-LEEG          PICTURE X
```

Als we ons richten op besturingsstructuren en identificatietekens, dan zijn we al in staat om het raamwerk van onze Procedure Division te schrijven. (De getallen in dit voorbeeld zullen dadelijk worden toegelicht.)

```
           PROCEDURE DIVISION.
           VERWERK-TABEL SECTION.
           VERW-TABEL.
               MOVE "O" TO EB-TABEL.
               MOVE "O" TO TABEL-LEEG.
1              READ TABEL
               AT END
                   MOVE "W" TO EB-TABEL
                   MOVE "W" TO TABEL-LEEG.
               MOVE "O" TO TABEL-ONGELDIG.
3, 4, 5        PERFORM VERWERK-TABELREC UNTIL EB-TABEL = "W"
               OR TABEL-ONGELDIG = "W".
2              IF TABEL-LEEG = "W"
                   DISPLAY "TABEL BESTAND LEEG"
               ELSE
                   IF TABEL-ONGELDIG = "W"
                       DISPLAY "TABEL BESTAND ONGELDIG"
                   ELSE
                       PERFORM VERWERK-MUT.
28             :
           VERWERK-TABELREC SECTION.
           VERW-TABELREC.
               :
           PRODUCEER-WS-TABELREC
6              :
               READ TABEL
               AT END
                   MOVE "W" TO EB-TABEL.
           VERWERK-MUT SECTION.
           VERW-MUT.
7              PERFORM PRODUCEER-KOP-OVERZICHT.                    13
19             MOVE "O" TO EB-MUT.
9              READ MUT
               AT END
                   MOVE "W" TO EB-MUT.
11             PERFORM VERWERK-MUTREC UNTIL EB-MUT = "W".
10,12          PERFORM PRODUCEER-VOETREGELS-OVERZICHT.
8              :
```

```
        VERWERK-MUTREC SECTION.
        VERW-MUTREC.
            MOVE "W" TO MUTREC-GELDIG.
17, 14      :
            IF MUTREC-GELDIG = "W"
                PERFORM VERWERK-GELDIG-MUTREC
                PERFORM PRODUCEER-GELDMUTREC
            ELSE
                PERFORM VERWERK-ONGELDIG-MUTREC
                PERFORM PRODUCEER-FOUTEN-GROEP.
            READ MUT
            AT END
                MOVE "W" TO EB-MUT.
        VERWERK-GELDIG-MUTREC SECTION.
            :
        PRODUCEER-GELDMUTREC SECTION.                          18
            :
        PRODUCEER-FOUTEN-GROEP SECTION.
        PROD-FOUTEN-GROEP.
20          PERFORM PRODUCEER-FOUT-REGEL-1.
            PERFORM PRODUCEER-FOUT-REGEL-2.
            PERFORM PRODUCEER-FOUT-REGEL-3.
        PRODUCEER-FOUT-REGEL-1 SECTION.
15, 21      :
        PRODUCEER-FOUT-REGEL-2 SECTION.                        22
            :
        PRODUCEER-FOUT-REGEL-3 SECTION.
16, 23      :
        PRODUCEER-VOETREGELS-OVERZICHT SECTION.
            PERFORM PRODUCEER-VOETREGEL-1.                     24
            PERFORM PRODUCEER-VOETREGEL-2.                     25
            PERFORM PRODUCEER-VOETREGEL-4.                     27
```

De volgende stap is het opstellen van een lijst van zoveel mogelijk zinnen, waarvan u verwacht dat ze in het programma moeten worden opgenomen en die niet al zijn behandeld. Het gaat hier om de volgende zinnen:

```
1    OPEN INPUT TABEL.
2    CLOSE TABEL.
3    MOVE 1 TO TABELSUB.
4    ADD 1 TO TABELSUB.
5    IF TABELSUB = 27 MOVE "W" TO TABEL-ONGELDIG.
6    MOVE TABELREC TO WS-TABELREC(TABELSUB).
7    OPEN OUTPUT FOUTEN-OVERZICHT.
8    CLOSE FOUTEN-OVERZICHT.
9    OPEN INPUT MUT.
10   CLOSE MUT.
11   OPEN OUTPUT GELDMUT.
12   CLOSE GELDMUT.
13   WRITE REGEL FROM KOP-OVERZICHT AFTER 2.
```

```
14  IF MUT-CODE NOT = "I" OR "W" OR "V"
        MOVE "*" TO EVT-BEG-KOL-2
        MOVE "0" TO MUTREC-GELDIG.
    en alle andere valideringszinnen.
15  MOVE MUT-CODE TO FOUTE-MUT-CODE
    en alle andere move-instructies die FOUTREGEL-1 produceren.
16  MOVE MUT-ADRES(3) TO FOUTREGEL-3.
17  MOVE SPACES TO FOUTREGEL-2.
18  WRITE GELDMUTREC FROM MUTREC.
19  MOVE 2 TO REGELTELLER.
20  ADD 4 TO REGELTELLER.
21  WRITE REGEL FROM FOUTREGEL-1 BEFORE 1.
22  WRITE REGEL FROM FOUTREGEL-2 BEFORE 1.
23  IF REGELTELLER LESS THAN 63
        WRITE REGEL FROM FOUTREGEL-3 BEFORE 2
    ELSE
        WRITE REGEL FROM FOUTREGEL-3 BEFORE PAGE.
    MOVE NUL TO REGELTELLER
24  WRITE REGEL FROM VOETREGEL-1 BEFORE 1.
25  WRITE REGEL FROM VOETREGEL-2 BEFORE 1.
26  WRITE REGEL FROM VOETREGEL-3 BEFORE 1.
27  WRITE REGEL FROM VOETREGEL-4 BEFORE 1.
28  STOP RUN.
29  MOVE MUT-ADRES(2) TO FOUTREGEL-ADR.
```

Uw zinnen zijn mogelijk anders geformuleerd, maar deze geringe variaties hebben verder geen betekenis. Of we nu BEFORE of AFTER na WRITE-instructies kiezen is volkomen arbitrair, maar als we eenmaal hebben gekozen moeten de andere instructies hierbij aansluiten om de gewenste spatiëring bij het afdrukken te krijgen.

Merk op dat in deze analyse de controle op de credietcode op dezelfde manier in een IF-instructie wordt afgehandeld als de controle op de MUT-code, zoals die in item 14 is weergegeven; deze controle houdt natuurlijk meer in, maar dat kunnen we voorlopig buiten beschouwing laten, als we ons nu maar bedenken dat een negatief resultaat van deze controle moet resulteren in het verplaatsen van een asterisk naar kolom 10 en in het op ongeldig zetten van het identificatieteken. Evenzo wordt de verplaatsing van de kredietlimiet voorlopig net zo afgehandeld als alle andere verplaatsingen van type 15.

Nu is het zaak deze zinnen (instructies) op de juiste plaats in het raamwerk te schikken. Als u een bepaalde zin wilt inpassen, vraag uzelf dan af: „Ik moet deze bewerking steeds doen wanneer ik....?" Het antwoord op deze vraag moet worden verschaft door een procedurenaam die u al heeft beschreven. Bijvoorbeeld:
zin 1 - Ik moet dit steeds doen wanneer ik....verwerk-tabel.
Zin 14 - Ik moet dit steeds doen wanneer ik....verwerk-mutrec.
Wanneer de procedure is gelokaliseerd, kan de zin gewoonlijk rechtstreeks in het raamwerk worden geplaatst. Bestanden worden gewoonlijk geopend bij de laatste mogelijkheid en gesloten bij de

eerste mogelijkheid. Probeer te vermijden dat de beginwaarden van parameters worden ontkoppeld van aanroepen naar subroutines. De nummers van de items (zinnen) zijn in de marges van het raamwerk geplaatst om te laten zien hoe deze zaak verder wordt voltooid.

De nummers die links van de instructies in het raamwerk staan, geven aan dat de desbetreffende zin vóór die regel moet worden ingevoegd. De nummers aan de rechterkant geven aan, dat de enkele zin alles omvat wat in de procedure aan de linkerkant aan de orde zal komen en die daarom kan vervangen.

De items 4 en 5 zijn bij item 3 geplaatst vanuit de overweging dat 2, 4 en 5 tesamen een PERFORM...VARYING...UNTIL constructie vormen. Zonder deze overweging, had item 4 aan het einde van VERWERK-TABELREC geplaatst moeten worden en item 5 aan het begin van VERWERK-TABELREC, waarbij regelingen getroffen zouden moeten worden voor het beëindigen van de procedure (exit), als de conditie waar zou zijn.

De beschrijving, het stellen van beginwaarden en het bijwerken van de totalen m.b.t. de voetregels zijn in het voorbeeld weggelaten; ze vormen een opgave voor de lezer.

Wanneer u enige ervaring hebt met de organisatie van programma's op de hier beschreven tamelijk mechanische wijze, bent u in een goede positie de meer directe benadering met gestructureerd Nederlands, zoals toegelicht in de volgende paragraaf, te proberen.

Opgave

1. Houd het raamwerk en de annotatie m.b.v. getallen uit deze paragraaf aan om het voorbeeld te herschrijven. U zult begrijpen dat TABELSUB = 27 hetzelfde betekent als TABEL-ONGELDIG = "W" en dat dit kan worden gebruikt om het speciale identificatieteken TABEL-ONGELDIG te elimineren. Doordat de eenmalige voorafgaande READ TABEL (waarin het structuurdiagram niet had voorzien) is ingevoerd kan ook het identificatieteken TABEL-LEEG worden verwijderd. Ga dit na.

   Als in een door een PERFORM aangeroepen paragraaf of section slechts één imperatieve instructie aanwezig is, moet u de PERFORM vervangen door die instructie. Een procedure zonder zinnen moet u verwijderen. Als het programma slechts één paragraaf per section blijkt te hebben, moet u de sections verwijderen.

L3 GESTRUCTUREERD NEDERLANDS EN DOE...STEEDS WANNEER

Wanneer men eenmaal inzicht heeft verworven in de essentie van gestructureerd programmeren door het gebruik van structuurdiagrammen, is het tamelijk gemakkelijk de programmastructuur weer te geven in gestructureerd Nederlands, zoals hieronder wordt toegelicht. Deze methode heeft het voordeel dat er veel gemakkelijker wijzigingen kunnen worden aangebracht dan bij diagrammen. Gestructureerd Nederlands kan top-down worden gebruikt door een systeemanalist, die de doeleinden van een programma wil specificeren, of analytisch door een programmeur die een programma wil ontwerpen.

Bij het schrijven van gestructureerd Nederlands is het het beste een vel papier met verticale lijnen te gebruiken (computerpapier een kwart slag gedraaid voldoet ook). Iedere lijn of marge wordt gebruikt om de reikwijdte van de gestructureerde constructies aan te geven. Volgordebewerkingen worden op achtereenvolgende regels tegen een marge aangeschreven. Instructies die ondergeschikt zijn aan een 'Indien' of een 'Anders' springen in, waarbij 'Anders' op één lijn staat met 'Indien', op de wijze zoals die in dit boek steeds is toegepast bij programmavoorbeelden. Maar er bestaat geen punt; de reikwijdte van een Indien of Anders wordt beperkt door de verschijning van een andere instructie verder weg met dezelfde marge, of door het fysieke einde van de procedure. Evenzo springen bij een herhaling de herhaalde instructies in vanaf een bepaalde marge en wordt de reikwijdte zichtbaar door een terugkeer naar die marge of door het fysieke einde van de procedure. (CODASYL heeft voorgesteld dat COBOL over expliciete afsluiters van de reikwijdte moet beschikken, zoals END-IF, END-PERFORM, waardoor COBOL-programma's kunnen worden gecodeerd op een bijna identieke wijze, zonder dat punten benodigd zijn.) Een subroutine kan worden uitgevoerd, in welk geval de procedurenaam van de subroutine wordt ingesloten in aanhalingstekens in de aanroepende procedure en wordt onderstreept aan het begin van de aangeroepen procedure. Laat tenminste één regel wit of het woord 'Einde' tussen de procedures.

Het volgende voorbeeld is het equivalent in gestructureerd Nederlands van de figuren 35 en 36.

Valideringsprogramma's

```
Verwerk TABEL om FOUTBOODSCHAP en WS-TABEL te produceren
Herhaal tot evb TABEL of tabel ongeldig
     |Verwerk TABELREC
     |Produceer WS-TABELREC
Als tabel leeg
     |Produceer BOODSCHAP-LEEG
Anders
     |Als tabel ongeldig
     |    |Produceer BOODSCHAP-ONGELDIG
     |Anders
     |    |Doe 'Verwerk MUT om GELDMUT en FOUTEN-OVERZICHT te
     |    |produceren'

Verwerk MUT om GELDMUT en FOUTEN-OVERZICHT te produceren
Produceer KOP-OVERZICHT
Verricht tot evb MUT
     |Verwerk MUTREC
     |Als MUTREC geldig
     |    |Verwerk GELDIG MUTREC
     |    |Produceer GELDMUTREC
     |Anders
     |    |Verwerk ONGELDIG MUTREC
     |    |Doe 'Produceer FOUTEN-GROEP'
Doe 'Produceer VOETREGELS-OVERZICHT'

Produceer FOUTEN-GROEP
Produceer FOUTREGEL-1
Produceer FOUTREGEL-2
Produceer FOUTREGEL-3

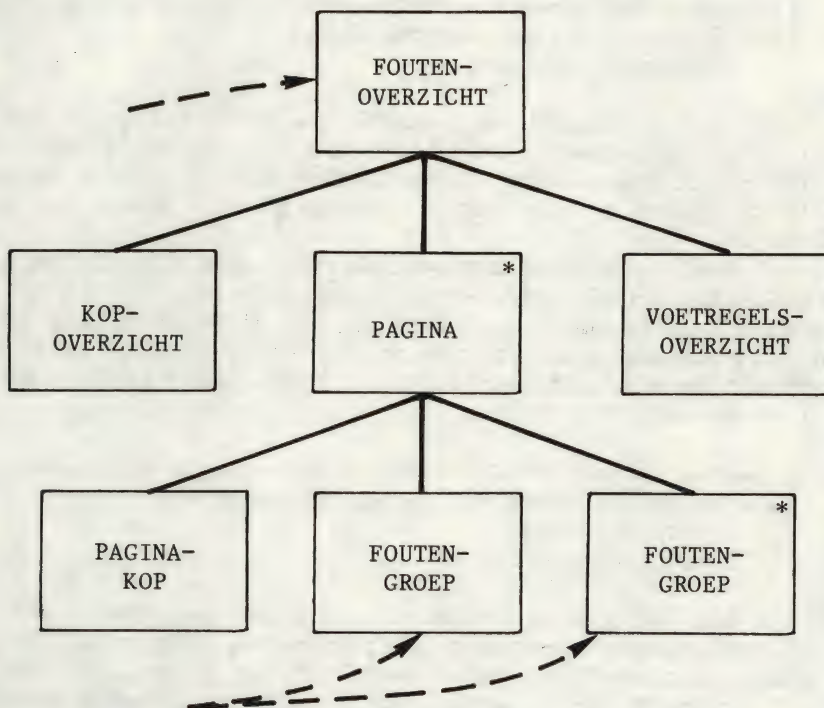
Produceer VOETREGELS-OVERZICHT
Produceer VOETREGEL-1
Produceer VOETREGEL-2
Produceer VOETREGEL-3
Produceer VOETREGEL-4
```

Deze beschrijving kan natuurlijk worden gebruikt om een raamwerkprogramma te produceren precies als hiervoor. Het zal duidelijk zijn dat gestructureerd Nederlands tot dezelfde organisatie van de tekst leidt als het structuurdiagram, behalve dat de 'keuze'-bewerkingen boven elkaar worden gestapeld i.p.v. naast elkaar.

Een veel voorkomende complicatie bij het ontwerpen van programma's met behulp van de methode uit paragraaf L2 is dat een uitvoerblok niet is afgekruist nadat het gestructureerde Nederlands is geproduceerd vanuit het gegevensstructuurdiagram. Dit probleem kan overwonnen worden d.m.v. een Doe...steeds wanneer...-constructie (deze wordt niet overal erkend als een constructie voor

gestructureerd programmeren). Van de Doe...Steeds wanneer kunt u zich een denkbeeld vormen door het niet afgekruiste blok te bekijken en de volgende vraag te stellen: „Ik wil deze uitvoer produceren steeds wanneer ik...?" Het geleverde antwoord dient een complete weergave te zijn van alle condities waaronder de uitvoer moet worden geproduceerd.

Stel dat een vereiste in het valideringsprobleem was een paginakop te produceren op iedere pagina van het FOUTEN-OVERZICHT waarin een FOUTEN-GROEP verscheen. De gegevensstructuur van FOUTEN-OVERZICHT kan dan worden gewijzigd zoals in figuur 37 is te zien.



FIGUUR 37 Herziene structuur van FOUTEN-OVERZICHT

We hebben nu twee blokken die de woorden FOUTEN-GROEP bevatten. De stippellijnen laten de 'gevorkte' pijl zien die komt van ONGELDIG MUTREC en geven aan dat een ONGELDIG MUTREC of met de eerste FOUTEN-GROEP in een PAGINA correspondeert of met een opvolgende FOUTEN-GROEP. Zo willen we dus nog steeds dat bij de verwerking van een ONGELDIG MUTREC een FOUTEN-GROEP wordt geproduceerd. Hetzelfde programmastructuurdiagram/gestructureerd Nederlands zal hieruit resulteren en beide blokken

met FOUTEN-GROEP kunnen worden afgekruist. Produceer KOP-OVERZICHT en Produceer VOETREGELS-OVERZICHT worden zoals tevoren behandeld. Produceer PAGINA en Produceer PAGINA-KOP blijven nu over en die kunnen we nu afhandelen door Doe...steeds wanneer constructies. „Ik wil PAGINA produceren steeds wanneer ...er een FOUTEN-GROEP moet worden geproduceerd en (het is de eerste FOUTEN-GROEP van het overzicht of ik heb de laatste regel van de eerdere FOUTEN-GROEP op minder dan vijf regels van het pagina-einde geschreven)! In gestructureerd Nederlands:

```
Steeds wanneer er een FOUTEN-GROEP moet worden geproduceerd
     |Als het de eerste FOUTEN-GROEP van het overzicht is of de
     |laatste regel van de eerdere FOUTEN-GROEP stond op minder
     |dan 5 regels van het pagina-einde
     |    |Produceer PAGINA
```

Het is duidelijk dat we bepaalde gegevens nodig hebben om de condities te herkennen - we moeten een identificatieteken EERSTE-FOUTEN-GROEP bedenken en we zouden REGELTELLER moeten bedenken, indien we dat niet al hadden gedaan. Kruis het blok PAGINA af.

We gaan verder met het overblijvende niet afgekruiste blok: „Ik wil PAGINA-KOP produceren steeds wanneer...ik PAGINA produceer". Met andere woorden: Produceer PAGINA-KOP is een bewerking die altijd plaatsvindt na Produceer PAGINA of een bewerking die een uitbreiding is van Produceer PAGINA. Dit voert tot deze conclusie:

```
Steeds wanneer er een FOUTEN-GROEP moet worden geproduceerd
     |Als eerste foutengroep of regelteller groter dan 61
     |    |Produceer PAGINA
     |    |Produceer PAGINA-KOP
```

Om een Doe...steeds wanneer... in het programma weer te geven moet u iedere plaats in het programma onderzoeken waar aan de 'steeds wanneer' conditie(s) wordt of kan worden voldaan en een zin invoegen die de logica van het gestructureerde Nederlands weergeeft. De invoering van identificatietekens of variabelen om een toetsbare conditie in het programma te creëren zal ook aanleiding geven tot andere zinnen (MOVE "W" TO EERSTE-FOUTEN-GROEP, MOVE "O" TO EERSTE-FOUTEN-GROEP, MOVE 0 TO REGELTELLER, ADD 1 TO REGELTELLER) die behandeld moeten worden volgens de reeds toegelichte wijze.

Het is duidelijk bij dit voorbeeld dat de 'steeds wanneer' in de procedure PRODUCEER-FOUTEN-GROEP hoort, waar we een openingszin als volgt nodig hebben:

```
IF EERSTE-FOUTEN-GROEP = "W" OR
 REGELTELLER GREATER THAN 61
    PERFORM PRODUCEER-PAGINA
    PERFORM PRODUCEER-PAGINA-KOP.
```

Na enige vereenvoudigingen kan deze codering er als volgt uitzien:

```
IF EERSTE-FOUTEN-GROEP = "W" OR
 REGELTELLER GREATER THAN 61
    MOVE "O" TO EERSTE-FOUTEN-GROEP
    MOVE ZERO TO REGELTELLER
    WRITE REGEL FROM PAGINA-KOP AFTER PAGE
    ADD 1 TO REGELTELLER.
```

Door deze aanpak kan de eerdere bewerking 23 vereenvoudigd worden tot WRITE REGEL FROM FOUTREGEL-3 BEFORE 2; en natuurlijk is er een andere duidelijke vereenvoudiging mogelijk in de codering hierboven (welke?).

Af en toe bestrijdt men vereenvoudiging van codering met het argument dat het ertoe leidt dat de koppeling tussen de programmastructuur en gegevensstructuur minder duidelijk wordt en als gevolg daarvan het programma minder gemakkelijk wordt onderhouden. Een tegenargument is dat een niet vereenvoudigd programma een grotere reikwijdte van de aandacht kan vereisen om begrepen te worden, hetgeen kan resulteren in fouten bij de programmeur.

De Doe...steeds wanneer... zorgt niet automatisch voor alle niet afgekruiste uitvoer. De uitvoervolgorde kan bijvoorbeeld tegenovergesteld zijn aan de invoervolgorde - misschien moet de invoer opnieuw worden gerangschikt. Wanneer er geen plaats in het programma is om de 'steeds wanneer'-logica in te voegen, of indien in de contekst van het tot zover ontwikkelde programma geen 'steeds wanneer' past, is een meer fundamentele wijziging noodzakelijk. Maar dit voert ons eerder naar het gebied van systeemontwerp i.p.v. naar dat van het ontwerpen van programma's.

Samengevat zijn de aanbevolen stappen bij het ontwerpen van programma's:

1. Teken gegevensstructuurdiagrammen en geef de samenhang aan.
2. Maak een specificatie in gestructureerd Nederlands op basis van het invoerbestand en de samenhang; voeg Doe...steeds wanneer... voor niet afgekruiste uitvoer toe.
3. Maak een raamwerk-programma en negeer daarbij de Doe... steeds wanneer...'s.
4. Voeg aan dit raamwerk alle zinnen toe die niet met 'steeds wanneer' hebben te maken.
5. Voeg 'steeds wanneer'-elementen en -zinnen toe.
6. Controleer en vereenvoudig indien dat helpt.

Als men over enige ervaring beschikt zal men geneigd zijn de specificatie in gestructureerd Nederlands direct op te schrijven. Toch is een ontwerp dat methodisch wordt afgeleid van het gegevensstructuurdiagram waarschijnlijk gemakkelijker te onderhouden.

Opgaven

1. Schrijf een specificatie in gestructureerd Nederlands van het bevolkingsexplosieprobleem. Doe dat
   a) d.m.v. afleiding uit de gegevensstructuren;
   b) aan de hand van figuur 25 (denk eraan de parameters te minimaliseren). Hoe verklaart u het verschil?

2. Schrijf een specificatie in gestructureerd Nederlands van het probleem m.b.t. de aanwezigheid van studenten (zie opgave 1, paragraaf L1).

3. Schrijf een specificatie in gestructureerd Nederlands van de opgave over het Gedicht (Appendix B).

L4 MEERVOUDIGE INVOERBESTANDEN

Wanneer er meerdere invoerbestanden in een probleem bestaan, moet u al de gegevensstructuren en samenhangen zoals hiervoor tekenen. Selecteer één bestand - het mutatiebestand - om de procedure te bepalen en maak de programmastructuur die resulteert uit de verwerking van dit bestand. Als er meerdere mutatiebestanden zijn, moet u het probleem behandelen alsof er twee programma's of twee fasen zijn. In de eerste fase worden dan de mutatiebestanden samengevoegd tot één mutatiebestand; in de tweede fase wordt dit samengevoegde mutatiebestand verwerkt om alle uitvoer te produceren.

Een typisch probleem is dat waarbij de verwerking van een mutatiebestand moet resulteren in het produceren van een overzicht, waaraan ook corresponderende stamrecords hun bijdrage moeten leveren. Er kunnen daarbij verschillende mutaties bestaan voor eenzelfde stamrecord, maar er is steeds slechts één stamrecord voor een gegeven mutatie. Hoe het stambestand moet worden behandeld, kan als volgt worden geschetst:

```
Steeds wanneer een mutatie moet worden verwerkt
    |Als de mutatie-identificatie niet gelijk is aan de
    |identificatie van het actuele stamrecord
    |    |Haal als actueel stamrecord op het stamrecord waarvan
    |    |de identificatie gelijk is aan de mutatie-identificatie
    |    |Als er niet zo'n stamrecord is
    |    |    |...
```

Als het stambestand sequentieel is georganiseerd met records gerangschikt naar opklimmende volgorde van identificatie en als de mutatierecords evenzo zijn gerangschikt, wordt bovenstaande schets aldus:

```
Steeds wanneer een mutatie moet worden verwerkt
        |Herhaal totdat identificatie van het stamrecord niet kleiner
        |dan mutatie-identificatie of eb stambestand
        |   |Lees stambestand
        |Als identificatie stamrecord groter dan mutatie-identificatie
        |of eb stambestand
        |   |...
```

Men zal inzien dat het in beide ontwerpen duidelijk is waar 'Verwerk STAMRECORD' geplaatst moet worden zodat we in het gegevensstructuurdiagram een invoerblok stamrecord kunnen afkruisen. Evenals bij mutatiebestanden wordt de weergave van de tweede versie in het programma bevorderd als één stamrecord wordt gelezen aan het begin van het programma; vervolgens wordt 'Verwerk STAMRECORD' geplaatst onder de besturing van de Herhaal...totdat..., waarbij de laatste actie van Verwerk STAMRECORD is: READ STAMRECORD AT END MOVE "W" TO EB-STAM.

Dit geval van een sequentieel georganiseerd invoer-stambestand dat wordt bijgewerkt door mutaties teneinde een sequentieel georganiseerd uitvoer-stambestand te produceren kan ook worden geanalyseerd met behulp van de 'steeds wanneer'-logica, bijvoorbeeld:

```
Steeds wanneer een stamrecord moet worden ingelezen (dat niet het
eerste record is)
        |Schrijf het actuele stamrecord weg
```

Dit geeft aanleiding ergens Produceer STAMUIT-RECORD te plaatsen.

Indien de mutaties ook betrekking kunnen hebben op het verwijderen van een stamrecord, zullen we iets als het volgende moeten insluiten:

```
Als het mutatietype verwijdering aangeeft
        |Verwijder het stamrecord
```

Wanneer het stamrecord wordt bijgewerkt door een nieuw bestand op te bouwen, wordt dit resultaat verkregen, nadat eenmaal is vastgelegd dat de actuele identificatie van het stamrecord gelijk is aan de actuele mutatie-identificatie, door het invoer-stamrecord over te slaan met een tweede 'read':

```
Als het mutatietype verwijdering aangeeft
        |Lees invoer-stambestand
```

Dit is een situatie waarbij het actuele stamrecord niet moet worden weggeschreven voordat een nieuw stamrecord wordt ingeingelezen; de 'steeds wanneer' hierboven moet worden gewijzigd:

Steeds wanneer een stamrecord moet worden ingelezen (dat niet het eerste record is) en het niet gaat om inlezen ter wille van een verwijdering
    |Schrijf het actuele stamrecord weg.

Het programma in Appendix A is een voorbeeld van bijwerken waarbij een nieuw bestand wordt opgebouwd.

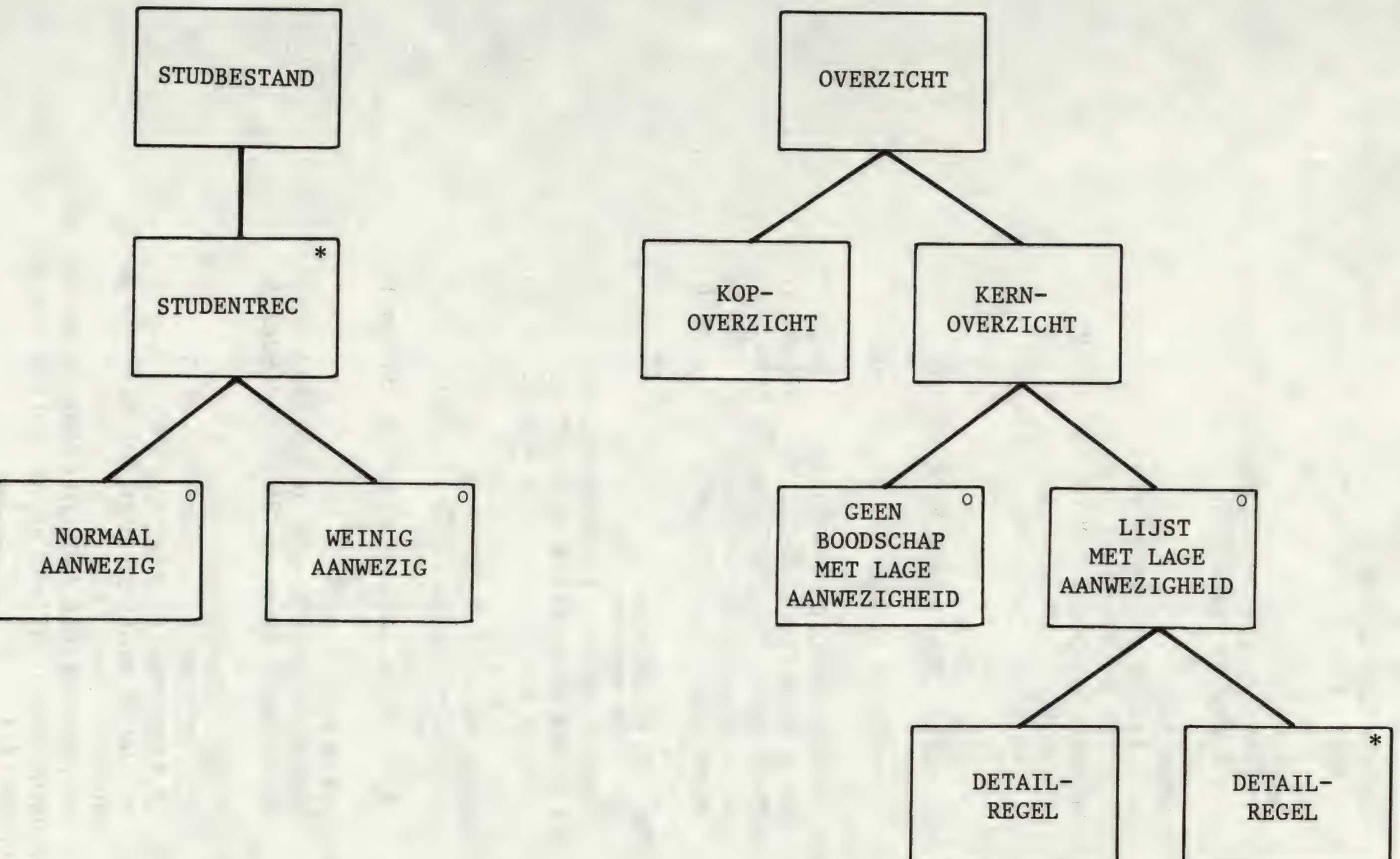
ANTWOORDEN HOOFDSTUK L

Paragraaf L1

1. Zie figuur 38.

Paragraaf L2

1.
```
PROCEDURE DIVISION.
VERWERK-TABEL.
    OPEN INPUT TABEL.
    READ TABEL
    AT END
        DISPLAY "TABELBESTAND LEEG"
        STOP RUN.
    PERFORM VERWERK-TABELREC VARYING TABELSUB FROM 1 BY 1
    UNTIL EB-TABEL = "W" OR TABELSUB GREATER THAN 26.
    CLOSE TABEL.
    IF TABELSUB GREATER THAN 26
        DISPLAY "TABELBESTAND ONGELDIG"
        STOP RUN.
    PERFORM VERWERK-MUT.
    STOP RUN.
*
VERWERK-TABELREC.
    MOVE TABELREC TO WS-TABELREC(TABELSUB).
    READ TABEL
    AT END
        MOVE "W" TO EB-TABEL.
*
VERWERK-MUT.
    OPEN OUTPUT FOUTEN-OVERZICHT.
    MOVE 2 TO REGELTELLER.
    WRITE REGEL FROM KOP-OVERZICHT BEFORE 2.
    OPEN INPUT MUT.
    MOVE "O" TO EB-MUT.
    READ MUT
    AT END
```



FIGUUR 38 Antwoord bij opgave 1, paragraaf L1

```
        MOVE "W" TO EB-MUT.
    OPEN OUTPUT GELDMUT.
    PERFORM VERWERK-MUTREC UNTIL EB-MUT = "W".
    CLOSE MUT.
    CLOSE GELDMUT.
    PERFORM PRODUCEER-VOETREGELS-OVERZICHT.
    CLOSE FOUTEN-OVERZICHT.
*
 VERWERK-MUTREC.
    MOVE SPACES TO FOUTREGEL-2.
    MOVE "W" TO MUTREC-GELDIG.
    IF MUT-CODE NOT = "I" OR "W" OR "V"
        MOVE "*" TO EVT-STER-KOL-2
        MOVE "O" TO MUTREC-GELDIG.
    :
    IF MUTREC-GELDIG = "W"
        WRITE GELDMUTREC FROM MUTREC
    ELSE
        PERFORM PRODUCEER-FOUTEN-GROEP.
*
 PRODUCEER-VOETREGELS-OVERZICHT.
    WRITE REGEL FROM VOETREGEL-1 BEGORE 1.
    WRITE REGEL FROM VOETREGEL-2 BEFORE 1.
    WRITE REGEL FROM VOETREGEL-3 BEFORE 1.
    WRITE REGEL FROM VOETREGEL-4 BEGORE 1.
*
 PRODUCEER-FOUTEN-GROEP.
    ADD 4 TO REGELTELLER.
    PERFORM PRODUCEER-FOUT-REGEL-1.
    PERFORM PRODUCEER-FOUT-REGEL-2.
    PERFORM PRODUCEER-FOUT-REGEL-3.
*
 PRODUCEER-FOUT-REGEL-1.
    MOVE MUT-CODE TO FOUTE-MUT-CODE.
    :
    WRITE REGEL FROM FOUTREGEL-1 BEFORE 1.
*
 PRODUCEER-FOUT-REGEL-2.
    MOVE MUT-ADRES(2) TO FOUTREGEL-2-ADR.
    WRITE REGEL FROM FOUTREGEL-2 BEFORE 1.
*
 PRODUCEER-FOUT-REGEL-3.
    MOVE MUT-ADRES(3) TO FOUTREGEL-3
    IF REGELTELLER LESS THAN 63
        WRITE REGEL FROM FOUTREGEL-3 BEFORE 3
    ELSE
        WRITE REGEL FROM FOUTREGEL-3 BEFORE PAGE.
```

(Wederom is het bijwerken van het totaal aan voetregels genegeerd.)

Paragraaf L3

1. a) Verwerk GROEIPERCENTAGES om OVERZICHT BEVOLKINGSEXPLOSIE te produceren

```
Tot einde van bestand van GROEPPERCENTAGES doen
     |Verwerk GROEIPERCENTAGE
     |DOEN: 'Produceer PAGINA MET RESULTATEN'
```

Produceer PAGINA MET RESULTATEN

```
Produceer GROEIPERCENTAGE REGEL
Herhaal, waarbij decade met 10 oploopt vanaf huidige
jaar + 10 totdat decade = huidige jaar + 210 of een size
error optreedt:
     |Produceer DETAIL REGEL
Doe 'Produceer FOUT'
```

Produceer FOUT

```
Als zich een size error heeft voorgedaan
     |Produceer SIZE ERROR REGEL
```

b) Bevolkingsexplosie

```
Bereken oppervlakte gebied
Tot einde van bestand doe:
     |Lees groeipercentage
     |Als niet einde van bestand
     |    |Doe 'Druk een pagina met resultaten af'
Stop run
```

Druk een pagina met resultaten af

```
Stel decade in op huidige jaar + 10, druk kopregels af
Bereken groeipercentage in decade
Stel bevolking aan het begin in
Tot decade = huidige jaar + 210 of een size error
optreedt, doe
     |Bereken nieuwe bevolking
     |Als zich een size error voordoet
     |    |Druk foutboodschap af
     |Anders
     |    |Bereken de nieuwe dichtheid
     |    |Als zich een size error voordoet
     |    |    |Druk fout-boodschap af
     |    |Anders
     |    |    |Druk een regel met resultaten af
     |Voeg 10 aan decade toe
```

Het verschil is dat a) een langs analytische weg verkregen ontwerp voor een raamwerkprogramma is waarin geen poging is gedaan om alle doeleinden te omvatten. Het komt overeen met het ontwerp van een programmeur, aan wie een ongestructureerde specificatie is overhandigd. Versie b) kan de

gestructureerde programmaspecificatie van de analist zijn - maar deze versie behoeft nog steeds enige uitbreiding om een volledige specificatie te zijn (bijvoorbeeld hoe moet de oppervlakte worden berekend?)

Aangezien de size error de herhaling in versie b) beëindigt, is er heel wat voor te zeggen dat de bewerking 'Voeg 10 aan decade toe' beter kan worden geplaatst onder de besturing van de laatste 'Anders'; er is geen noodzaak de decade aan te passen als zich een fout heeft voorgedaan. Maar omdat de status van 'decade' niet is voorgeschreven als zich een fout voordoet, is het niet echt fout zoals het er nu staat en natuurlijk sluit het nauwer aan bij PERFORM... VARYING...UNTIL.

2 en 3

Haal een collega om uw antwoord te controleren.

M DOCUMENTATIE

M1 PROGRAMMADOCUMENTATIE

Twee hoofddocumenten laten zien wat een programma in detail doet:

de programmaspecificatie, met bestands- en recordbeschrijvingen, met goed-gestructureerde beschrijvingen in het Nederlands van de werking van procedures en met beslissingstabellen of een gelijkwaardig mechanisme om de bedoeling van een programma weer te geven en
de programma-uitlijsting.

Het eerste is een *ontwerp*document: dit wordt gemaakt voordat het coderen begint. Het wordt vaak door de systeemanalist en niet door de programmeur gemaakt. Hoewel bijna altijd wordt geëist dat zulke documenten worden bijgewerkt om de huidige staat van het programma weer te geven, wordt dit bijwerken dikwijls 'na de gebeurtenis' gedaan, waardoor er verschil kan bestaan tussen dit document en de uitlijsting. De uitlijsting is daarom het enige werkelijk betrouwbare document van wat het programma presteert en programmeurs wenden zich dan ook tot dit document wanneer zij een wijziging of verbetering van programmadetails overwegen.

Daarom moet de frontlijn van documentatie de programma-uitlijsting zijn. Niet alleen is de uitlijsting de vertrouwenwekkendste van de beschikbare documenten, maar hij wordt bovendien relatief moeiteloos en automatisch bijgewerkt door de programmeur.

Hoewel het gemakkelijk is programmadocumentatie te rechtvaardigen als hulpmiddel bij latere programmawijzigingen, is er een ander motief dat nog meer kracht heeft, maar minder voor de hand ligt. Een programmeur die op een consistente wijze zorgt voor documentatie, zoals we hier beschreven hebben, verkeert in een werksituatie die veel minder ontvankelijk is voor fouten dan zijn collega die niet of slecht documenteert. Aangezien het vaak onmogelijk is een programma na het ontwerp ervan volledig te testen, is het van wezenlijk belang dat de ontwikkelingsmethodiek kwaliteit bezit. De documentatie is een hulpmiddel om het doel steeds helder in het oog te blijven houden bij het weergeven van het probleem in programmacode.

Voor kleine tot middelgrote programma's bestaat er een zeer goede strategie voor de ontwikkeling van programma's. Deze bestaat uit drie fasen:

1. Stel een gedetailleerde beschrijving in gestructureerd Nederlands op en laat een collega deze lezen en in details goedkeuren.
2. Codeer systematisch op basis van het gestructureerde Nederlands en laat een collega deze codering lezen en in details goedkeuren.
3. Test het programma systematisch, zoals beschreven zal worden in hoofdstuk N en laat een collega de omvang van de test en de testresultaten zien en goedkeuren.

Bij grote programma's kan het wenselijk zijn een in gestructureerd Nederlands gestelde specificatie op een nog hoger niveau te maken, waarbij misschien programma's worden beschreven die afzonderlijk kunnen worden gecompileerd en getest om daarna aan elkaar gekoppeld te worden (hoofdstuk U). Dit moet ook worden gecontroleerd voordat er een systematische uitwerking in meer gedetailleerde specificaties van plaatsvindt.

Een recht-toe-recht-aan geschreven COBOL-programma levert echter geen uitlijsting op die adequaat voldoet aan deze extra *verklarende* behoeften die we eraan willen opleggen. De volgende paragrafen houden zich bezig met aanwijzingen om de uitlijsting zo te wijzigen dat aan deze behoeften tegemoet wordt gekomen.

M2 DOCUMENTATIE IN DE IDENTIFICATION DIVISION

Het is aan te bevelen de volgende informatie aan het begin van een programma op te nemen, zodat die ook op de uitlijsting verschijnt:

1. de plaats van dit programma in de reeks
2. de globale doelen van het programma
3. het globale ontwerp van het programma
4. de logica van het programma in gewone taal.

Als er een goede standaard voor systeemdocumentatie van kracht is (zie bijvoorbeeld hoofdstukken 6 en 12 van *Systems Analysis*, uitgegeven door Edward Arnold in de UK), kan voor de eerste twee worden volstaan met verwijzingen naar andere documenten. Zo niet, dan kunnen we als volgt een eenvoudig verslag opnemen:

```
*INVOER      - ARTIKEL MUTATIEBESTAND "ARTMUT" GEPRODUCEERD
*              DOOR PROGRAMMA PAR002VET
*            - OUD ARTIKEL STAMBESTAND "ARTSTAM" EERDER
*              UITGEVOERD DOOR DIT PROGRAMMA
```

```
*UITVOER    - NIEUW ARTIKEL STAMBESTAND "ARTSTAM"
*           - VERANDERINGENBESTAND "ARTWIJZ", DAT VERDER
*             WORDT VERWERKT DOOR PROGRAMMA PAR004AFDRUK
*VERWERKING - HET ARTMUT BESTAND BEVAT WIJZIGINGEN,
*             VERWIJDERINGEN EN INVOEGINGEN EN IS
*             GESORTEERD NAAR VOLGORDE VAN ARTIKELNUMMER.
*             HET GESORTEERDE BESTAND WORDT GEBRUIKT OM
*             HET OUDE ARTIKELSTAMBESTAND BIJ TE WERKEN
*             MET ALS RESULTAAT EEN NIEUW ARTIKELSTAM-
*             BESTAND. FOUTEN TIJDENS HET BIJWERKEN VER-
*             OORZAAKT DOORDAT TRANSACTIERECORDS NIET OVER-
*             EENSTEMMEN MET RECORDS VAN HET STAMBESTAND
*             WORDEN UITGELIJST OP DE REGELDRUKKER, TERWIJL
*             GEACCEPTEERDE TRANSACTIES WORDEN VASTGELEGD
*             IN HET VERANDERINGENBESTAND, TEZAMEN MET DE
*             NIEUWE STAAT VAN HET STAMRECORD. DEZE GEGE-
*             VENS WORDEN IN EEN VOLGEND PROGRAMMA AFGE-
*             DRUKT. EEN CONTROLETELLING VAN DE GEWIJZIGDE,
*             VERWIJDERDE EN INGEVOEGDE RECORDS WORDT BIJ-
*             GEHOUDEN, EVENALS EEN TELLING VAN DE RECORDS
*             IN HET OUDE EN NIEUWE STAMBESTAND. DEZE
*             TOTALEN WORDEN AFGEDRUKT BIJ HET EINDE VAN
*             DE RUN.
```

Aan de laatste twee behoeften: de weergave van de globale logica en het ontwerp van het programma, kan worden voldaan door de specificatie in gestructureerd Nederlands op te nemen.

Als een gedeelte van de logica is beschreven door beslissingstabellen of statustabellen (zie hoofdstuk P), neemt men deze gewoonlijk ook in de uitlijsting op.

Wanneer het gehele proces is samengesteld uit afzonderlijk gecompileerde programma's (hoofdstuk U), behoren de naam en het doel van ieder programma uitgelijst te worden in het hoofdprogramma. Een ander hulpmiddel is te laten zien hoe het programma is georganiseerd in een hiërarchie van paragrafen en sections – zie paragraaf M4.

Opgaven

1. De programmeur kan de compiler instrueren om naar de kop van een nieuwe pagina van de uitlijsting te gaan door een schuine streep "/" in kolom 7 te plaatsen. Waarom is dit zinvol wanneer de Identification Division op de hier beschreven wijze van commentaar moet worden voorzien?

2. De meeste automatiseringsafdelingen hebben een standaardmethode voor het vaststellen van programmanamen ingevoerd. De bedoeling hiervan is (a) te verzekeren dat ieder programma een eigen naam heeft en (b) het systeem waartoe het programma behoort te identificeren. Waarom?

3. Wat is de praktische waarde van het opnemen van de paragrafen AUTHOR en DATE-COMPILED?

M3 DOCUMENTATIE IN DE DATA DIVISION

Documentatie in de Data Division concentreert zich rondom *naamafspraken* en *aanvullende informatie*.

Naamafspraken hebben twee doeleinden:

1. Bevorderen van de plaatsbepaling van een in de Data Division aanwezige beschrijving van een specifieke gegevensnaam wanneer deze in de Procedure Division wordt tegengekomen, bijvoorbeeld om te kunnen zien wat de PICTURE is.
2. Betekenis geven aan een gegeven.

Een plezierig en logisch stelsel dat beide doeleinden verwezenlijkt laat iedere gegevensnaam door een voorvoegsel van twee letters voorafgaan. Voor kleine programma's is dit wat overdreven, maar als programma's groter worden begint het zijn nut af te werpen. De eerste letter laat zien in welke section van de Data Division het element wordt beschreven:

F = File Section
R = Report Section
W = Working-Storage Section, etc.

De tweede letter van het voorvoegsel wordt gebruikt om het record waarin het gegevenselement staat te identificeren. De records worden van een letter voorzien in de volgorde waarin ze worden gecodeerd. Een volledig voorbeeld staat in Appendix A.

De hier bedoelde naamafspraken veronderstellen dat de programmeur vrijheid van handelen heeft om de gegevenselementen van een naam te voorzien. In een goed-georganiseerde afdeling gegevensverwerking kunnen de systeemanalisten een woordenboek of lijst van alle in het systeem gebruikte gegevens hebben aangelegd. Ze kunnen verder de namen van alle bestanden, records en gegevenselementen hebben beschreven. Als dit het geval is, dient de programmeur dezelfde namen in de File Section te gebruiken. Als er dubbele namen ontstaan als gevolg van deze werkwijze (bijvoorbeeld wanneer het wijzigen van een sequentieel stambestand het opbouwen van een nieuw stambestand betekent), is een aanvaardbare oplossing: het toevoegen van een achtervoegsel -I of -U als onderscheiding tussen invoer- en uitvoergegevenselementen.

In de Working-Storage Section zullen sommige records natuurlijk in samenhang met een invoer- of uitvoerrecordtype ontstaan. De overblijvende gegevenselementen moeten door de programmeur worden georganiseerd in records. De volgende adviezen zullen dit

proces vergemakkelijken:

1. Zet in een gemeenschappelijk record al die gegevenselementen waarvan de bedoeling is dat ze globaal toegankelijk zijn in het programma, dat wil zeggen elementen die worden gebruikt in de hoofdprocedure en subroutines.
2. Zet in een gemeenschappelijk record al die gegevenselementen die als parameters worden meegegeven bij de aanroep van een subroutine (dat wil zeggen één record voor iedere PERFORM die een of meer parameters meegeeft).
3. Zet in een gemeenschappelijk record al die gegevenselementen die plaatselijk in een subroutine horen, dat wil zeggen alleen binnen de subroutine worden gebruikt (één record per subroutine met één of meer plaatselijke gegevenselementen).

Aan de gegevens wordt betekenis gegeven door de keuze van goede namen, hetgeen grotendeels een kwestie van gezond verstand is. In het algemeen worden de gegevensnamen hierdoor langer, maar het is toch zonneklaar dat zelfs VERSCHULDIGDE-DATUM-VAN-VOLGENDE-BETALING moet worden verkozen boven VDVVB.

Een extra voordeel van het gebruik van de voorvoegsels is dat het per ongeluk gebruiken van gereserveerde woorden wordt vermeden. Een extra voordeel van het gebruik van zinvolle namen is dat foutieve spelling in de Procedure Division minder waarschijnlijk is.

Aanvullende informatie moet het invullen van informatieleemtes die COBOL openlaat bevorderen. De beschrijvingen in de Data Division kunnen ons wel van een element zeggen:

1. zijn omvang en type (PICTURE)
2. zijn beginwaarde (VALUE)
3. zijn interne geheugenformaat (USAGE)
4. zijn doel (als we een zinvolle naam hebben bedacht)
5. de betekenis van zijn waarde als het een indicator of code is (als u conditienamen gebruikt, zie § Q4)
6. waardoor het wordt geïndiceerd (subscripted) als het om een tabel gaat (als u INDEXED BY gebruikt, zie § O5).

Wat de COBOL-beschrijvingen ons niet zeggen is:

1. zijn doel (als dit niet volledig uit de naam is af te lezen)
2. de reeks waarden die het kan hebben (als deze kleiner is dan het totale bereik dat door de PICTURE wordt aangegeven)
3. de betekenis van zijn waarden als het om een indicator of code gaat en u geen conditienamen hebt gebruikt
4. waardoor het wordt geïndiceerd (subscripted) als het om een tabel gaat (tenzij de INDEXED BY-passage is gebruikt).

Deze aanvullende informatie kan worden gegeven in een commentaar dat onmiddellijk volgt op de gegevensbeschrijving. Het raamwerk voor dit commentaar kan zijn:

```
*doel
*BEREIK reeks van waarden
*code BETEKENT betekenis (zo vaak als nodig is)
*GEINDICEERD DOOR indexnaam.
```

De programmeur behoort het schrijven van dit commentaar te overwegen na iedere gegevensbeschrijving waarbij hij natuurlijk de onderdelen die niet van toepassing of overbodig zijn weglaat. Als bijvoorbeeld het doel van het element duidelijk is op grond van zijn naam, als het de gehele reeks waarden kan aannemen die door zijn picture worden geïmpliceerd, als het geen code of indicator is, als er geen geldigheidstesten mee worden uitgevoerd en als het geen tabel is, kan het gehele commentaar achterwege blijven. Het voorbeeldprogramma in Appendix A laat deze methode in de praktijk zien. Het enige commentaar dat daarin nodig is betreft WA-FOUTMELDINGEN waarvan het gebruik iets complexer is dan normaal. Als er een data dictionary in gebruik is waarin het doel, het bereik van gegevenselementen etc. staan vermeld, hoeft deze informatie natuurlijk niet ook nog eens in het programma opgenomen te worden.

Opgave

1. Stel de gegevensbeschrijving en het commentaar op voor een tabel van 12 namen van maanden van 3 tekens in working storage, die moet worden gebruikt om een numerieke maand te vertalen, waarna deze vertaling op de kop van een formulier wordt afgedrukt.

M4 DOCUMENTATIE IN DE PROCEDURE DIVISION

De belangrijkste aanwijzingen voor het leesgemak zijn:

1. schrijf één opdracht per regel
2. geef de paragraafnaam een eigen regel
3. volg de afspraken voor het inspringen bij conditionele opdrachten op
4. als het nesten meer dan 3 à 4 niveaus diep gaat, overweeg in plaats hiervan dan het gebruik van een vooruitspringende GO TO of een out-of-line subroutine; of neem commentaar op waarin de analyse, die aanleiding gaf tot de zo diep geneste opdrachten, wordt weergegeven
5. gebruik lege regels om de presentatie te verbeteren, bijvoorbeeld om blokken van paragrafen die logisch met elkaar zijn verbonden te identificeren
6. gebruik paragraaafnamen die de procedures in de paragraaf beschrijven

7. gebruik een voorvoegsel bij een procedurenaam om de plaats van de procedure te helpen bepalen.

Een goed voorvoegselsysteem voor procedures in een gestructureerd programma maakt gebruik van een voorvoegsel van twee tekens. Het eerst teken is een letter die het niveau aangeeft waarop de paragraaf, het blok met paragrafen of de section wordt aangeroepen. De hoofdprocedure staat op niveau A en deze voert met behulp van PERFORM procedures op niveau B uit. Een procedure op niveau C wordt aangeroepen door een procedure op niveau B, enz. Het tweede teken is een rangordegetal dat samen met het eerste teken de procedure uniek identificeert en tevens helpt de procedure te lokaliseren in een grote uitlijsting. De procedures moeten worden gecodeerd in de uitlijsting op volgorde van hun voorvoegsels. Als een procedureblok bestaat uit een section of groep van paragrafen, dienen alle paragrafen en dat blok hetzelfde voorvoegsel te krijgen.

Zo zal in een programma, waarin geen enkele procedure opnieuw wordt gebruikt op een andere plaats, de hiërarchie van de programmabesturing altijd volgens deze lijnen verlopen:

```
A
    B1
        C1
        :
        Cn
    B2
        Cn+1
        :
        Cn+m
    B3
        Cn+m+1
        :
    B4
    :
```

Wanneer een procedure wel opnieuw wordt gebruikt, adviseren we om deze te coderen op een niveau dat overeenkomt met het gebruik op het hoogste niveau ervan. Het voorvoegsel dat aan de procedure zou worden toegekend bij gebruik in het andere geval, moet worden weggelaten. Dit klinkt een tikje ingewikkeld, maar ik weet zeker dat u, als u het probeert, het een handige oplossing vindt.

Tegen het hernieuwd gebruik van procedures bestaat het volgende argument. Een opnieuw gebruikte procedure wordt in tenminste twee verschillende contexten gebruikt. Als de procedure is gewijzigd in één van deze contexten, kan de wijziging deze procedure verbasteren in de andere context. Procedures behoren òf te worden gedupliceerd, òf de documentatie behoort het hernieuwde gebruik duidelijk te maken.

De hiërarchie van de besturing moet worden gedocumenteerd in het commentaar in de Identification Section om het programma-onderhoud te vergemakkelijken. Dit kan worden bereikt door het eenvoudig uitlijsten van de namen van procedureblokken en deze te laten inspringen overeenkomstig hun niveau. Het is aan te bevelen de voorvoegsels, die zijn weggelaten bij de procedures in verband met hernieuwd gebruik, nu wel op te nemen met een verwijzing naar de plaats waar de codering van deze procedure staat. Het programma in Appendix A geeft een voorbeeld van deze aanpak.

Opgave

1. Leid voorvoegsels voor de procedurenamen af uit het programma dat als antwoord hoort bij opgave 1 van paragraaf L2.

ANTWOORDEN HOOFDSTUK M

Paragraaf M2

1. De uitlijsting kan zodanig worden ingericht dat ieder belangrijk nieuw onderwerp op een nieuwe pagina begint. Ook kan het splitsen van een beslissingstabel of een specificatie in gestructureerd Nederlands over twee pagina's worden vermeden.
2. (a) Ieder programma moet een eigen naam hebben zodat het kan worden geplaatst in en opgehaald uit de bibliotheek met doeltaalprogramma's; zie hoofdstuk U.
   (b) De andere programma's die tot het systeem behoren kunnen gemakkelijk worden geïdentificeerd en gelokaliseerd als de uitlijstingen worden gerangschikt in volgorde van programmanaam. Dit kan nuttig zijn bij het begrijpen van de programma-uitlijsting of het volgen van de consequenties van een wijziging.
3. AUTHOR: Het kan handig zijn te weten wie de oorspronkelijke programmeur is als het programma verkeerd loopt of wijziging behoeft.
   DATE-COMPILED: Als meer dan één versie van de uitlijsting is geregistreerd, kan dit de actuele versie identificeren.

Paragraaf M3

1.
```
    03 WD-TABEL-INHOUD VALUE etc.
       05 WD-ALPHA-MAAND OCCURS 12 TIMES PICTURE XXX.
*         WORDT GEBRUIKT OM EEN NUMERIEK WEERGEGEVEN MAAND OM
*         TE ZETTEN T.B.V. DE KOP VAN HET OVERZICHT;
*         GEINDICEERD DOOR FC-NUMERIEKE-MAAND
```

Paragraaf M4

```
1.    A1-VERWERK-TABEL
         B1-VERWERK-TABELREC
         B2-VERWERK-MUT
            C1-VERWERK-MUTREC
               D1-PRODUCEER-FOUTEN-GROEP
                  E1-PRODUCEER-FOUT-REGEL-1
                  E2-PRODUCEER-FOUT-REGEL-2
                  E3-PRODUCEER-FOUT-REGEL-3
            C2-PRODUCEER-VOETREGELS-OVERZICHT
```

N HET TESTEN VAN PROGRAMMA'S

N1 DOEL VAN FUNCTIONEEL TESTEN

Het testen van computerprocedures kan vanuit drie gezichtspunten worden bekeken: dat van de gebruiker, de analist en de programmeur.

De gebruiker wil vaststellen dat het programma goed aansluit op de eisen van zijn organisatie. De gebruiker is de enige die uiteindelijk kan beoordelen of het programma nu de gewenste resultaten produceert, omdat hij de enige is (naar ik aanneem) die afdoend kan beslissen wat de 'gewenste resultaten' zijn.

De systeemanalist heeft getracht de wensen van de gebruiker te bevatten, maar hij kan hierin tekort zijn geschoten. Hij zal ernaar streven vast te stellen, dat het programma overeenstemt met de specificatie die hij de gebruiker heeft helpen produceren en dat het programma functioneert zoals was bedoeld, wanneer het wordt gedraaid in een reeks andere programma's en menselijke procedures.

De programmeur heeft getracht de wensen van de systeemanalist en gebruiker te bevatten, maar hij kan hierin tekort zijn geschoten. Hij zal ernaar streven vast te stellen dat het programma functioneert in overeenstemming met zijn interpretatie van de specificatie. Als in de fase van het testen van het programma wordt ontdekt dat de specificatie niet klopt, is het zeer veel duurder de fout te corrigeren dan als deze was ontdekt ten tijde van de specificatie.

Deze drie partijen kunnen ook verschillende standpunten hebben, wanneer het erop aankomt de testgegevens te formuleren. De testgegevens worden bepaald om hypothetische fouten uit het programma te halen; iedere partij heeft verschillende beweegredenen voor deze hypotheses, zoals hierna blijkt.

De gebruiker beschouwt het totale informatiesysteem als een 'black box'. Zijn keuze van testgegevens spruit geheel voort uit zijn begrip van wat belangrijk is voor hem en de organisatie. Hij kiest bijvoorbeeld gegevens betreffende zijn belangrijkste klant. Hoewel de programmeur geen enkele reden kan bedenken, waarom zijn programma zich bij de belangrijkste klant anders dan voor iedere andere klant zou gedragen, behoren de testgegevens van de gebruiker hierom niet te worden afgewezen. Zijn gegevens testen waarschijnlijk de situatie waarbij het grootste verlies zou kunnen optreden en dit

alleen is al een rechtvaardiging. De gebruiker kan andere geldige redenen hebben bij de selectie van testgegevens, bijvoorbeeld: hij herinnert zich een gebeurtenis die verleden jaar moeilijkheden bezorgde bij het oude systeem en hij vindt dat dezelfde transactie moet worden uitgetest in het nieuwe systeem.

De systeemanalist kan het programma ook als een 'black box' beschouwen, maar hij beschikt waarschijnlijk over een algemeen inzicht in computerprogramma's en hij kan een ander soort fout veronderstellen dan de fouten die de gebruiker of de programmeur zich indenken. Hij weet bijvoorbeeld dat nulhoeveelheden of -bedragen of lege velden vaak niet worden verwerkt zoals bedoeld is, of dat soms de bedoeling in dergelijke situaties niet helder uit de specificatie voortvloeit. Door de systeemanalist bedachte testgegevens kunnen een waardevolle toevoeging zijn aan die van de gebruiker en programmeur.

De programmeur beschouwt het testen vanuit een volledige kennis van de bewerkingen van zijn programma. Aangezien hij de exacte structuur van het programma kent, verkeert hij in een positie om fouten te veronderstellen die zouden ontsnappen aan de aandacht van de analist en de gebruiker; bijvoorbeeld: overloop in de variabelen van working storage, overloop van tabellen, foutief rekenkundig teken. De testgegevens van de programmeur moeten de soliditeit van het programma onderzoeken door alle uiterste situaties na te gaan. Als het correct werkt in de grensgevallen, is het zeer waarschijnlijk (vooral bij programma's in de praktijk) dat het programma ook werkt voor alle gevallen daartussen. De gebruiker dient hierin te participeren door de correctheid van de resultaten te bekrachtigen, zelfs van resultaten die geproduceerd werden door testgegevens van de programmeur.

Het doel van functioneel testen is het vertrouwen in de functionele correctheid van het programma te verbeteren. Het is duidelijk dat alle drie partijen testgegevens moeten overleggen; de testgegevens van de programmeur vormen slechts één stap op de weg naar dit vertrouwen. De programmeur dient zich toe te leggen op het ontwerpen van tests in overeenstemming met zijn speciale kennis van de programmastructuur (en als de programmeur tegelijk ook de analist of gebruiker is, moet hij bovendien proberen te denken als analist of gebruiker). Iedere test moet een nieuw facet van het programma onderzoeken. Als de programmeur erin slaagt iedere test iets anders te laten controleren, zal hij een extra motief hebben om alle resultaten te verifiëren.

Het testen van een programma is een noodzakelijk, maar niet voldoende middel om vertrouwen erin te kweken. Een typisch programma uit de praktijk heeft al gauw te maken met een paar honderdduizend verschillende mogelijke uitvoerwaarden en het is volstrekt onmogelijk die allemaal te gaan testen. Even belangrijk als het vertrouwen dat voortvloeit uit het testen is het vertrouwen dat ontstaat als gevolg van het gebruik van een solide methode van programmaontwerp (zie de hoofdstukken K en L), als gevolg van bureaucontrole (desk-checking) en als gevolg van nauwgezette controle door een

collega of opdrachtgever van ontwerp en codering (zie paragraaf M1).

Er zijn formele methoden ontwikkeld om de correctheid van programma's te bewijzen, maar de toepassing van deze methoden is moeilijk en kostbaar in andere dan eenvoudige situaties. Bij afwezigheid van een formeel bewijs is een nauwgezette bureaucontrole van de programmabewerkingen vereist om informeel overtuigd te raken van de correctheid van het programma. Ervaren programmeurs hebben er slag van te herkennen, welke delen van het programma niet helemaal helder zijn en zijn in staat mentaal het programma met testgegevens te traceren, waardoor ze de correctheid kunnen overzien.

Er is een gezonde psychologische basis voor de veronderstelling dat een gedetailleerde programmacontrole door een collega fouten aan het licht kan brengen die aan de aandacht van de oorspronkelijke programmeur waren ontsnapt. Deze veronderstelling wordt ook gesteund door experimenteel onderzoek. De oorspronkelijke programmeur raakt gefixeerd op één bepaalde interpretatie van het programma, waarvan hij moeilijk afstand kan nemen. Een nieuwe geest kent niet dezelfde beperking.

Opgave

1. Wanneer een test een fout in een programma aan het licht brengt en u herstelt die fout, is uw vertrouwen in het programma dan toegenomen of juist verminderd?

N2 MINIMAAL-GRONDIG TESTEN

De programmeur kan een programma niet 'volledig' testen, alsof het een black box zou zijn, aangezien dat zou inhouden dat iedere mogelijke combinatie van invoergegevenswaarden in iedere mogelijke opeenvolging aan het programma zou moeten worden aangeboden. Zelfs voor heel gewone programma's is dit een enorm aantal.

Er is een streefniveau van testen vereist dat in de praktijk kan worden gehaald en een redelijke mate van betrouwbaarheid verschaft: een 'grondige' test. Hoewel ik de voorkeur eraan geef niet exact aan te geven wat wordt bedoeld met een 'grondige' test, kan ik wel een *minimaal-grondige* test definiëren en andere testmogelijkheden voor een bepaald programma naar voren brengen. Een grondige test kan dan vrij vaag worden gedefinieerd als een minimaal-grondige test opgesteld door de programmeur tezamen met een selectie van tests opgesteld door de programmeur, analist en gebruiker.

Een minimaal-grondige test is een test zodanig dat iedere enkelvoudige conditie in het programma wordt uitgevoerd in beide waarden ervan (waar en onwaar) waar dat mogelijk is. 'Waar dat mogelijk is' is vereist, omdat er in het programma condities kunnen zijn opge-

nomen waarvan één der waarden niet kan optreden bij welke invoergegevens dan ook. De oorzaak van deze onmogelijkheid kan òf een logische tegenstrijdigheid zijn òf het gevolg zijn van de praktische onmogelijkheid om de vereiste hoeveelheid gegevens aan te leveren. Een voorbeeld is te zien in paragraaf K4, waar een defensieve SIZE ERROR-passage werd toegevoegd aan een COMPUTE-instructie. Er is geen bekende invoerwaarde die de size error zal veroorzaken; de SIZE ERROR-passage vloeit voort uit een soort harnas-filosofie.

Een soortgelijke situatie treedt een enkele maal op wanneer de programmeur een logische fout heeft gemaakt, waardoor een stukje van de codering niet te bereiken is; een poging om minimaal-grondig te testen zal dergelijke fouten aan het licht brengen. Een andere situatie doet zich voor, wanneer een programma velden totaliseert van records waarvan het aantal dat aanwezig is in een bestand niet is gespecificeerd. De programmeur kan de omvang van het totaalveld zodanig instellen dat hij denkt dat de mogelijkheid van het optreden van een size error in de praktijk is uitgesloten. Maar omdat er geen omvang van het totaal-veld bestaat die de theoretische mogelijkheid van overloop uitsluit, is het terecht om een SIZE ERROR-passage op te nemen, in hoofdzaak om eventuele logische fouten op te sporen. De conditie: size error true die het gevolg is van het opnemen van deze passage kan ook worden gezien als voortvloeiende uit een onmogelijke verzameling invoergegevens. Minimaal-grondig testen, zoals hierboven gedefinieerd, houdt geen rekening met dergelijke uitzonderlijke situaties, maar garandeert dat er geen enkele instructie in het programma is die niet minimaal één keer tijdens het testen wordt uitgevoerd. Merk op dat 'enkelvoudige conditie' in de definitie conditionele passages zoals SIZE ERROR evenals condities in IF-zinnen omvat.

Als eenzelfde conditie op meer dan één plaats verschijnt, moet ieder vóórkomen daarvan worden getest. We kunnen de verzameling testsituaties creëren door een T of een O, die aangeeft wat de waarde is, te plaatsen bij iedere defensieve conditie waarvoor geen test kan worden gecreëerd. Specificeer vervolgens testinvoer en plaats een T of een O bij iedere daardoor geteste conditie om de poot aan te tonen die de test de computer zal doen nemen. Let nu op iedere conditie die nog niet zowel een T als een O heeft en specificeer een nieuwe test die de conditie zal bereiken en tot gevolg heeft dat de ongeteste poot wordt genomen. Plaats nu opnieuw een T of een O bij alle niet al zo aangeduide condities die nu zijn beproefd door de tests. Ga hiermee door totdat alle condities zowel een T als een O hebben. Leg bij iedere gecreëerde testsituatie uw voorspelling van het resultaat vast.

Soms kan een conditie in een programma niet worden getest zonder meer testgegevens dan u op dit ogenblik van plan was in te voeren, bijvoorbeeld pagina-overloop op een grote pagina. Er kan dan een interim-test worden uitgevoerd door een passende wijziging van de conditie, bijvoorbeeld door het aantal regels per pagina te verminderen. Toch moet het programma worden getest op de eigenlijke conditie, voordat het uiteindelijk voor produktiedoeleinden wordt geaccepteerd.

Opgaven

1. Specificeer minimaal-grondige tests voor de eerste section van het valideringsprogramma waarvan de Procedure Division wordt geschetst in paragraaf L2 (de opmaak van de records wordt gegeven in paragraaf J3). Een leeg MUT-bestand zal volstaan voor deze opgave.

2. Hoe kunt u het bedenken van records van 800 tekens vermijden tijdens de allereerste testfase van het GEDICHT-programma (Appendix B)?

N3 ANDERE TESTS

Een andere gedachte betreffende het opzetten van testsituaties, waarbij men de programmastructuur kent, bestaat in het onderzoeken van het programma met minimum- en maximumwaarden voor de invoergegevens en minimum- en maximumwaarden bij het testen van de condities. Het argument hierachter is dat het programma, als het functioneert bij deze extreme waarden, waarschijnlijk ook functioneert voor alle situaties daartussen, omdat de meeste programma's in de praktijk een regelmatige of monotone (technisch gesproken) verwerking kennen.

In de praktijk is het moeilijk deze gedachte tot het uiterste door te voeren en bovendien blijkt het doorvoeren ervan niet altijd het vertrouwen te vergroten. Bijvoorbeeld: size errors hebben gewoonlijk een abnormale beëindiging in programma's in de praktijk tot gevolg. Als dat zo is, lijkt het niet zo zinnig de size error condities te testen met waarden die nu juist alleen maar een size error veroorzaken en met waarden die de grootst mogelijke size error verroorzaken. Toch blijkt het een tamelijk algemene regel te zijn om het gebruik van maximum/minimumwaarden in overweging te nemen en iedere situatie op zichzelf te beoordelen.

Om extreme testsituaties te produceren moet u ieder numeriek veld in de invoerrecordindelingen onderzoeken. Maak dan tests waarin ieder van deze numerieke velden de kleinste en grootste waarden krijgt die volgens de definitie mogelijk zijn. Dikwijls zullen dit de waarden zijn die toegestaan worden door de PICTURE van het element (bijvoorbeeld A PIC S99; kleinste waarde van A is -99, grootste waarde van A is +99). Soms worden resultaten als 'onbepaald' gespecificeerd, indien invoerwaarden buiten een zeker gebied gaan (bijvoorbeeld doordat een eerder programma heeft gevalideerd of de gegevens binnen dat gebied blijven). In dit geval moet u de kleinste en grootste waarde kiezen in het gebied waarvoor de resultaten zijn gedefinieerd. Onderzoek nu iedere vergelijkingsconditie voor numerieke gegevens in het programma en overtuig u ervan dat

testsituaties worden opgenomen die aanleiding geven tot de kleinste en grootste waarde die de conditie waar maken en de kleinste en grootste waarde die de conditie onwaar maken (bijvoorbeeld IF A GREATER THAN 50; testsituaties voor A = -99, A = 50, A = 51, A = +99). Wanneer de vergelijkingsconditie 'gelijkheid' inhoudt, bijvoorbeeld IF A = 50, moet er een testsituatie zijn waarbij A = 50; het ongelijke resultaat kan worden getest met behulp van minimum- en maximumwaarden van A.

Wanneer de conditie een lusbeëindiger (bijvoorbeeld een teller) is, bestaat er gewoonlijk slechts één waarde die kan worden bedacht om de conditie te beëindigen. Het zij zo.

Andere soorten tests om in overweging te nemen zijn die waarbij de volgende fouten aan het licht kunnen komen:

1. Nulfouten. Testgegevens waarin de numerieke invoer nul is, moeten worden ingesloten. Bekijk ook niet-nul zijnde testgegevens, die een nulresultaat opleveren, bijvoorbeeld transacties die een rekeningsaldo van nul opleveren.
2. Overloopfouten. Eventuele overloop van numerieke velden, strings, tabellen of bestandsruimte.
3. Fouten met betrekking tot lege bestanden. Als het besturingssysteem de invoer van bestanden zonder records toestaat, handelt het programma deze situatie dan correct af?
4. Fouten met betrekking tot de opeenvolging van situaties. Als de behandeling van het ene invoerrecord afhankelijk is van hoe het vorige record werd verwerkt, of als verschillende recordtypes bestaan in de invoer, bekijk dan de omzettingen.
5. Fouten waarbij het bestand niet past. Bijvoorbeeld: een mutatie waarvoor geen stam aanwezig is.
6. Einde-van-bestand fouten. Vooral wanneer het bijwerken van een stambestand het opbouwen van een nieuw stambestand inhoudt, dient een test te worden opgenomen om te garanderen dat invoegingen vóór het begin van de oude stam en na het einde van de oude stam evenals invoegingen binnen de oude stam correct worden afgehandeld. Gelijke aandacht moet worden gegeven aan verwijderingen, vooral een verwijdering van het laatste record in de oude stam. Dezelfde soort filosofie kan van toepassing zijn op sommige bewerkingen met tabellen.
7. Afrondingsfouten. Vooral van belang bij iedere soort berekening van samengestelde interest, omdat onvoldoende nauwkeurigheid in working storage-variabelen kan leiden tot een fout in een betekenisvol cijfer van het resultaat. De test moet de langste periode van samenstelling beslaan en de resultaten moeten met de hand worden gecontroleerd of met een calculator (maar pas op: vele goedkope calculators zijn onnauwkeurig, wanneer nauwkeurigheid van vele cijfers vereist is).
8. Cyclusfouten. Wanneer een uitvoerbestand opnieuw moet worden ingevoerd in hetzelfde programma bij de volgende uitvoering, moet deze cyclus minimaal één keer worden getest.

Opgave

1. Bedenk de testgegevens voor het bevolkingsexplosie-programma (Appendix B).

N4 ZICHZELF-CONTROLERENDE PROGRAMMA'S

Een extra methode is het programma zo te ontwerpen dat het zichzelf controleert. Het idee is de verkrijging van de resultaten door middel van twee verschillende methoden als een controle op de programmalogica. Dit is hetzelfde idee dat accountants kennen als 'kruistellingen'. Hoe dit wordt gedaan hangt sterk af van de precieze aard van het programma; hier zijn een paar voorbeelden.

Veronderstel dat een mutatiebestand kan toevoegen aan of verwijderen uit een stambestand. Steeds wanneer het programma een mutatie leest, voegt het 1 toe aan een mutatieteller, en het voegt eveneens, steeds wanneer het een stambestand leest of schrijft, 1 toe aan een oude of nieuwe stamteller. Op het punt in het programma waar het bepaalt dat de mutatie een toevoeging is, voegt het 1 toe aan een toevoegingsteller, terwijl het in het andere geval 1 toevoegt aan een verwijderingsteller. Aan het einde van het programma controleert het of:

mutatieteller = toevoegingsteller + verwijderingsteller

en of:

nieuwe stamteller = oude stamteller + toevoegingsteller - verwijderingsteller.

Het is belangrijk dat er geen vertakkingsopdrachten worden geschreven tussen de opdracht die de teller ophoogt en de bewerking die geteld wordt.

Een ander voorbeeld: we stellen ons een programma voor dat records leest die een waarde en een code bevatten en dat de totalen van de waarden per code berekent plus een totaal-generaal. We moeten de waarde aan het totaal-generaal toevoegen zodra het record is gelezen (dat wil zeggen geen vertakkingsopdrachten tussen de READ- en deze ADD-opdracht – dit kan betekenen dat we de ADD-opdracht conditioneel maken ten opzichte van de READ-opdracht wanneer die niet het einde van het bestand heeft ontdekt), en dan vervolgen met het toevoegen van de waarde aan het erbij passende totaal per code. Aan het einde van het programma moet het alle totalen sommeren en controleren of deze overeenkomen met het totaal-generaal. (Deze specifieke controle zou evenwel weinig zin hebben, als u gebruik maakt van hogere voorzieningen, zoals de SUM-passage beschreven in paragraaf S3.)

Dit alles mag u overdreven voorkomen, omdat u vindt dat bij een eenmaal getest programma zichzelf-controlerende procedures overbodig zijn. Maar bedenk dat programma's in de praktijk in het

algemeen een lang bestaan hebben en regelmatig worden herzien en onderhouden. Uw controleroutine functioneert tevens als een bevestiging dat toekomstige wijzigingen bevredigend zijn aangebracht.

ANTWOORDEN HOOFDSTUK N

Paragraaf N1

1. Rationeel gesproken kunt u in beide richtingen beïnvloed zijn. De veiligste filosofie is uw vertrouwen in het programma te *verminderen* bij iedere fout die u vindt. Als u meer dan twee of drie fouten in een programma aantreft, moet u overwegen het vanaf het nulpunt te herschrijven en daarbij een solide methode van programma-ontwerp te hanteren.

   Misschien verrast dit antwoord u. Ik ken zeker ook een aantal programmeurs die de omgekeerde filosofie aanhangen. De kwestie is dat er een enorm aantal mogelijkheden is waarvan wordt verondersteld dat ze correct door het programma worden verwerkt. Ieder aantal mogelijkheden zou foutief verwerkt kunnen worden, maar uw testgegevens kunnen slechts een paar daarvan beoordelen. Voordat u gaat testen, heeft u een zekere mate van vertrouwen (waarschijnlijkheidsoordeel) dat deze niet-geteste mogelijkheden correct worden verwerkt. Iedere door uw programma goed *doorlopen* test draagt er toe bij te bevestigen dat het programma goed in elkaar zit en zal uw vertrouwen dat de niet-geteste mogelijkheden correct worden verwerkt doen toenemen. Omgekeerd: iedere *mislukte* test draagt er toe bij te bevestigen dat het programma niet deugt en behoort uw vertrouwen dat de niet-geteste mogelijkheden correct worden verwerkt, te doen afnemen.

   Dit afgenomen vertrouwen kan gedeeltelijk of geheel worden opgeheven door het toegenomen vertrouwen dat verkregen moet worden, wanneer de fout is hersteld. Maar er bestaat een reële kans dat de programmeur zijn vertrouwen als gevolg hiervan meer dan gerechtvaardigd is laat toenemen. Daarom denk ik dat het beter is aan de voorzichtige kant te blijven en dit aspect te negeren.

Paragraaf N2

1. De eenvoudige condities in de eerste section zijn (READ TABEL) AT END; EB-TABEL = "W"; TABEL-ONGELDIG = "W"; TABEL-LEEG = "W"; TABEL-ONGELDIG = "W" (tweede keer). Geen onmogelijke waarden. Tests gaan als volgt.
   Eerste run. Leeg bestand TABEL.

Resultaten: AT END waar; EB-TABEL = "W" waar; TABEL-LEEG = "W" onwaar.
Voorspeld resultaat: boodschap "TABEL BESTAND LEEG".
Tweede run. Het bestand TABEL heeft één record bevattende A9999999 (de gegevens zijn willekeurig). MUT bestand leeg.
Niet eerder geteste resultaten: AT END onwaar; EB-TABEL = "W" onwaar; TABEL-ONGELDIG = "W" onwaar; TABEL-LEEG = "W" onwaar; TABEL-ONGELDIG = "W" (tweede keer) onwaar.
Voorspeld resultaat: overzicht met kop- en voetregels; geen ongeldige records, nullen in alle totalen-generaal.
Derde run. Tabelbestand heeft 27 records met willekeurige gegevens.
Niet eerder geteste resultaten: TABEL-ONGELDIG = "W" waar; TABEL-ONGELDIG = "W" (tweede keer) waar.
Voorspeld resultaat: boodschap "TABEL BESTAND ONGELDIG".

2. Reduceer de recordomvang tot een passender maat, bijvoorbeeld 80 tekens. Reduceer de regellengte tot bijvoorbeeld 60 tekens om te voorkomen dat het programma alleen de bijzondere situatie, waarbij de invoerrecordomvang gelijk was aan de uitvoerrecordomvang, afhandelt. Een programma dat op een bevredigende wijze 80 invoertekens en 60 uitvoertekens behandelt mag worden geacht zich goed te gedragen, wanneer deze constanten worden gewijzigd in respectievelijk 800 en 80; maar deze verwachting behoeft natuurlijk nog wel een laatste test.

Paragraaf N3

1. Test 1 : Leeg bestand.
   Test 2 : Drie records.
   0000
   1001 (of ieder ander gegeven dat het minst betekenisvolle cijfer gebruikt; voorspel de uiteindelijke bevolking en dichtheid)
   9999 (verwachte size error)
   Beloon uzelf met een hoog cijfer indien u concludeerde dat één van de size errors (in òf bevolking òf dichtheid) onmogelijk moet zijn.

O TECHNIEKEN VOOR HET WERKEN MET TABELLEN EN GEGEVENS

O1 INSPECT

De opdracht INSPECT kan worden gebruikt om bewerkingen uit te voeren met de individuele tekens in een gegevenselement.

INSPECT identifier-1 TALLYING identifier-2 FOR {ALL / LEADING} literal-1

Veronderstel dat we een veld van zes cijfers hebben met als naam NUM en als inhoud ____27. Indien we het aantal beginspaties willen bepalen, kunnen we een gegevenselement, laten we zeggen SPATIE-AANTAL, definiëren en als volgt het resultaat verkrijgen

```
MOVE ZERO TO SPATIE-AANTAL.
INSPECT NUM TALLYING SPATIE-AANTAL FOR LEADING SPACES.
```

De INSPECT-opdracht telt het gevonden aantal (in dit geval het aantal spaties) op bij identifier-2; de programmeur moet dit gegevenselement dus de juiste beginwaarde geven.

De ALL-variatie levert ons een telling van alle literal-1's in het veld op, of ze nu aan het begin staan of niet. De BEFORE INITIAL-variatie (zie Appendix D) telt het aantal tekens op dat aan literal-1 voorafgaat. Wanneer we bijvoorbeeld een veld ACHTERNAAM hebben met een lengte van 20 tekens, dat iemands achternaam gevolgd door spaties bevat, en we willen vastleggen hoe lang de achternaam is, dan kunnen we schrijven:

```
MOVE ZERO TO SPATIE-AANTAL.
INSPECT ACHTERNAAM TALLYING SPATIE-AANTAL
        FOR CHARACTERS BEFORE INITIAL SPACE.
```

Het antwoord zal wederom in SPATIE-AANTAL staan.

Bij gebruikmaking van de keuzemogelijkheid REPLACING (Appendix D), wordt iedere literal-3 vervangen door literal-4. Indien NUM ____27 bevat en we coderen

```
MOVE ZERO TO SPATIE-AANTAL.
INSPECT NUM TALLYING LEADING SPACES
        REPLACING LEADING SPACES BY ZEROS.
```

dan zal na afloop SPATIE-AANTAL 4 bevatten en NUM 000027.

Tellingen en vervangingen kunnen zowel voor als na een bijzondere waarde in het desbetreffende veld worden uitgevoerd; het aantal keren dat een bepaalde reeks tekens in het veld voorkomt (in tegenstelling tot één enkel teken) kan door de tally worden bijgehouden of vervangen door een andere reeks; meer dan één bewerking met tally of replacing tegelijkertijd is mogelijk. Ik ben niet voornemens al deze alternatieven te behandelen, maar ik wil wel een speciaal voorbeeld ontvouwen als inleiding op grondgedachten die hierna volgen.

Veronderstel dat we een grote hoeveelheid gegevens ontvangen die door een andere computer dan de onze zijn geregistreerd. Computers van verschillende fabrikanten (of zelfs verschillende computertypes van dezelfde fabrikant) kunnen verschillende interne codes gebruiken voor de weergave van de tekens. Soms is die weergave voor bijna alle tekens verschillend; vaker bestaat de weergave voor de meeste tekens uit dezelfde code afgezien van een paar bijzondere tekens waarvoor hij verschilt.

Laten we aannemen dat de gegevens van de andere computer een $-teken bevatten, maar dat de code voor $ op de andere machine de code voor £ op onze machine is. Wanneer onze computer de gegevens leest, zullen alle $-tekens ten onrechte worden gelezen als £-tekens. Onze taak bestaat in de vervanging van de foutieve £-tekens door $-tekens in de code van onze computer; op deze wijze beschikken we over de correcte gegevens.

We kunnen deze taak volbrengen met

```
INSPECT INVOER-RECORD REPLACING ALL "£" BY "$".
```

Maar een voordeel van de INSPECT-opdracht is de mogelijkheid meer dan één teken te vertalen door één opdracht. Veronderstel dat ons probleem er als volgt uitziet:

Gegevens van andere computer	Ingelezen door onze computer als
$	£
*	!
?	*
!	(
)	?

We kunnen deze gehele vertalingstabel voor de code in een enkele INSPECT-opdracht opnemen:

```
INSPECT INVOER RECORD
REPLACING ALL "£" BY "$", "!" BY "*",
              "*" BY "?", "(" BY "!",
              "?" BY ")".
```

(Als een ingrijpende vertaling van gegevens van een geheel afwijkende computer nodig is, kan het beter zijn de CODE-SET voorzieningen van COBOL, zo die beschikbaar zijn, toe te passen; raadpleeg hiervoor uw manual. CODASYL heeft ook een uitbreiding voorgesteld, nl. de INSPECT ... CONVERTING ..., die voor dit soort gevallen mogelijk beter geschikt is.)

Opgave

1. U moet een record UITVOER-REC wegschrijven dat zal worden ingelezen door een andere machine. De codes van de twee machines zijn dezelfde op de volgende uitzonderingen na:

Uitvoer van uw machine	Ingelezen door andere machine
;	.
.	;
*	=
=	/
/	*
#	)
&	(
)	#
(	&

Wijzig de gegevens in UITVOER-REC zo, dat ze correct worden ingelezen door de andere machine. Deze opgave lijkt eenvoudiger dan hij is.

O2 TABELRAADPLEGING

We zijn de grondgedachte van een tabel al tegengekomen. Hij bestaat uit herhaalde gegevenselementen die in een OCCURS-passage worden beschreven.

Laten we het probleem van opgave 1 in paragraaf O1 als voorbeeld nemen. De codes die worden uitgevoerd door uw machine zijn de *doel*codes; de codes in de oorspronkelijke gegevens zijn de *bron*-codes. Met het oog op de vertaling van de broncodes in de doelcodes kunnen we de relevante programmagedeeltes als volgt neerschrijven:

```
01 REGEL.
   02 TEKEN PICTURE X OCCURS 100 TIMES.
WORKING-STORAGE SECTION.
01 TABELLEN.
   02 BRONTABEL VALUE ".;=/*)(#&".
      03 BRONTEKEN PICTURE X OCCURS 9 TIMES.
   02 DOELTABEL VALUE ";.*=/#&)(".
      03 DOELTEKEN PICTURE X OCCURS 9 TIMES.
01 INDICES COMP SYNC RIGHT.
   02 TEKENTELLER PICTURE 999.
   02 TABELTELLER PICTURE 99.
```

```
PROCEDURE DIVISION.
:
(opdrachten die REGEL van zijn inhoud voorzien)
   PERFORM B1-REGEL-VERTALEN.
:
B1-REGEL-VERTALEN.
   PERFORM C1-TEKEN-VERTALEN VARYING
   TEKENTELLER FROM 1 BY 1 UNTIL TEKENTELLER = 101.
C1-TEKEN-VERTALEN.
   PERFORM D1-VERGELIJKEN VARYING TABELTELLER
   FROM 1 BY 1 UNTIL TEKEN(TEKENTELLER) =
   BRONTEKEN(TABELTELLER) OR TABELTELLER = 10.
   IF TABELTELLER LESS THAN 10
      MOVE DOELTEKEN(TABELTELLER) TO TEKEN(TEKENTELLER).
D1-VERGELIJKEN.
```

D1-VERGELIJKEN is een dummy-paragraaf waarin geen enkele actie plaatsvindt.

Het eerste teken van REGEL wordt vergeleken met het eerste, tweede, derde, etc. teken in BRONTABEL. Als er geen gelijk teken wordt gevonden (en dat houdt in dat TABELTELLER de waarde 10 heeft bereikt), hoeft het teken niet vertaald te worden. Wordt er wel een gelijk teken aangetroffen dan wordt het corresponderende teken uit DOELTABEL er voor in de plaats gezet. Daarna wordt het volgende teken in REGEL bekeken onder de besturing van de eerste PERFORM ... VARYING ... en vindt herhaling van het proces plaats totdat alle tekens zijn vertaald.

Er bestaan andere methodes om hetzelfde doel te bereiken. Op sommige machines bijvoorbeeld kan de interne code van het te vertalen teken in plaats daarvan worden behandeld als een gelijkwaardig (binair) getal. Aangezien elke code verschillend is, zullen ze ieder aanleiding geven tot een enig getal, dat direct kan worden gebruikt als een index om de doelcode uit een tabel op te halen (de doeltabel zal de complete tekenverzameling moeten bevatten).

In paragraaf O5 wordt behandeld hoe tabelraadpleging kan plaatsvinden zonder een dummy-paragraaf, indien de SEARCH-opdracht beschikbaar is.

Opgave

1. (Nogal moeilijk) U leest kaarten die werden geponst op een machine die de eerste negen letters van het alfabet heeft gecodeerd in uw code 1 t/m 9 en omgekeerd. Bovendien betekent de code die u leest als * een reeks van drie nullen op de andere machine. Het record dat u inleest heeft een lengte van 80 tekens, maar deze kan toenemen tot 240 tekens na vertaling (dat wil zeggen als er alleen maar * zijn). Het vertaalde record moet worden ingebouwd in een record van 240 tekens in working

storage met als naam NIEUW-REC; de oorspronkelijke kaart noemen we IN-REC.
Schrijf de code om de vertaling tot stand te brengen.

O3 TABELORGANISATIE

Wanneer in een grote tabel naar elementen wordt gezocht en er moeten veel elementen worden gevonden, is het duidelijk dat er heel wat verwerkingstijd in dat zoekproces gaat zitten. Veronderstel dat we 50.000 records, die ieder voor zich 1000 tekens bevatten, uit een andere in onze eigen computercode moeten vertalen. Bij gebruik van het tabelonderzoek zoals beschreven in paragraaf O2 zouden in totaal 50.000 × 1.000 = 50 miljoen tabelraadplegingen moeten worden gemaakt. Als de tabel een lengte van 60 tekens zou hebben en ieder teken evenveel kans in de invoergegevens te staan, dan zouden gemiddeld 30 tekens in de tabel moeten worden onderzocht voordat het goede teken was gevonden. Er zouden in totaal 30 × 50 miljoen vergelijkingen moeten worden gemaakt; zelfs op een snelle moderne computer zou de hoeveelheid tijd benodigd om 1500 miljoen vergelijkingen uit te voeren eerder in uren dan minuten moeten worden gemeten. Wanneer het om een grote tabel gaat, kan het daarom wenselijk zijn snellere zoekmethoden te vinden.

Wanneer de tabelgegevens achter elkaar worden onderzocht van links naar rechts, spreken we van een *sequentieel onderzoek*. Een handige manier om een sequentieel onderzoek te versnellen is het achter elkaar plaatsen van de tabelelementen van links naar rechts in een dalende frequentie van benodigdheid. Wanneer we de elementen die het vaakst voorkomen aan het begin van de tabel plaatsen, zal dat ertoe leiden dat gemiddeld minder elementen worden onderzocht voordat het benodigde element is gevonden.

Veronderstel bijvoorbeeld dat in het hierboven genoemde geval twee van de te vertalen elementen in de invoer nul en spatie zijn en dat 40% van alle invoertekens nul en 30% spaties zijn. Wanneer we nul en spatie op de eerste en tweede plaats in de tabel zetten, betekent dat het volgende:

40% van de zoekbewerkingen vereist 1 vergelijking
30% van de zoekbewerkingen vereist 2 vergelijkingen
30% van de zoekbewerkingen vereist $2 + \frac{60 - 2}{2}$ = gemiddeld 31 vergelijkingen

Het totaalgemiddelde aantal vergelijkingen zou dus zijn:

$$\frac{40 \times 1 + 30 \times 2 + 30 \times 31}{100} = 10.3$$

Er is geen opgave bij deze paragraaf, maar u zou nog eens de laatste alinea van paragraaf O2 kunnen lezen, waar de zichzelf-indicerende methode van tabelraadpleging kort wordt beschreven. Waar deze techniek kan worden gebruikt zijn geen vergelijkingen nodig.

O4 DE BINAIRE OF LOGARITMISCHE ZOEKMETHODE

Dit is een zoekmethode die, als alle elementen in gelijke frequentie benodigd zijn, zo min mogelijk vergelijkingen garandeert (wannneer zelf-indicerend niet mogelijk is).

De tabelelementen worden gerangschikt in een opklimmende waardenreeks, bijvoorbeeld:

BE	ED	EP	HO	LI	OM	VA	VE

FIGUUR 39 Tabel 1 (Gemeentecodes met twee tekens).

Laten we aannemen dat een gemeentecode is ingelezen bij de invoergegevens en vertaald moet worden in de gemeentenaam die in een andere tabel wordt gegeven:

BESTΔ	EDEΔΔ	EPEΔΔ	HOORN	LISSE	OMMEN	VAALS	VENLO

FIGUUR 40 Tabel 2 (Gemeentenamen met vijf tekens).

Laat het aantal tabelelementen N zijn (= 8 in tabel 1). Onderzoek het element N/2 (= 4) vanaf het begin. Aangenomen dat dit niet het gewenste element is, gaat de zoekmethode als volgt verder: als het gewenste element *groter* is dan het pas onderzochte elemtent, houdt dit feitelijk in dat de hele linkerkant van de tabel onbelangrijk is. Onderzoek de (N/4)-de (= 2^e) elementpositie *rechts* van het laatst onderzochte element.

Laten we aannemen dat het gewenste element *kleiner* is dan dit nieuwe element uit de tabel; dit betekent dat het gehele rechter kwart van de tabel onbelangrijk is. Daarom onderzoeken we de (N/8)-ste (= eerste) positie *links* van het laatst onderzochte element. Op deze wijze wordt de plaats van het element bepaald.

In het algemeen zien we dus dat de eerste vergelijking de halve tabel elimineert, de tweede vergelijking opnieuw een kwart, de derde een achtste, enz. Het maximale aantal vergelijkingen benodigd om een element in een tabel van N elementen te lokaliseren is ${}^2\log(N)+1$, terwijl het gemiddelde aantal minder dan dit is.

Opgave

1. (Nogal moeilijk) Een kaart bevat een tweecijferige plaatscode in PL-CODE die als volgt moet worden vertaald:

Code	Plaats	Code	Plaats
MO	MOKUM	KO	KOOG
AS	ASSEN	EP	EPE
SN	SNEEK	OM	OMMEN
BE	BEST	VA	VAALS
ED	EDE	HO	HOORN
DE	DELFT	VE	VENLO
LA	LAREN	LI	LISSE

Neem aan dat de code op de kaart correct is.
Teken dan een programmastroomschema dat een binaire zoekmethode ter vertaling van deze code uitvoert. Tracht het proces ook in gestructureerd Nederlands te beschrijven. In de praktijk zal men voor zo'n probleem als dit nogal eens zowel een programmastroomschema als een specificatie in gestructureerd Nederlands maken om na te gaan of de beschrijving van de procedure correct is. Test uw stroomschema met code LI en één of twee andere codes om er zeker van te zijn dat het werkt (u kunt een afkappingsprobleem hebben wanneer het aantal elementen in de tabel niet een macht van 2 is; een manier om dit probleem te ontlopen bestaat hierin dat we het aantal elementen tot een macht van 2 maken en de grootst mogelijke COBOL-waarde in de overtollige elementen plaatsen).
Schrijf ook de COBOL-opdrachten benodigd om de zoekmethode uit te voeren. (Het is mogelijk in dit voorbeeld met slechts één tabel te werken.)

O5 SEARCH

Grotere compilers voorzien ook in een automatische zoekmethode voor een 1-dimensionale tabel. Er zijn een aantal keuzemogelijkheden beschikbaar, maar de methode verloopt in wezen als volgt.

1. De naam van de index ('index-naam') die moet worden gebruikt bij de tabel wordt gedeclareerd door een INDEXED-BY-passage in de gegevensbeschrijving voor de tabel, bijvoorbeeld:

```
01 TABEL.
   02 TABEL-ELEMENT PICTURE XX OCCURS 100 TIMES
      INDEXED BY TAB-INDEX.
```

De gespecificeerde index-naam (TAB-INDEX in het voorbeeld) wordt verder nergens in de Data Division beschreven; de compiler voorziet zelf hierin. De index verandert automatisch van waarde wanneer een SEARCH plaatsvindt.

2. De gewenste beginwaarde van de index wordt bereikt door de SET-opdracht. Deze gewenste waarde is gewoonlijk 1 om er zeker van te zijn dat het zoeken in de tabel begint bij het eerste tabelelement, bijvoorbeeld:

```
SET TAB-INDEX TO 1.
```

3. De tabel wordt doorzocht door middel van de opdracht met het formaat:

```
SEARCH identifier AT END imperative-statement-1

WHEN condition-1 {imperative-statement-2}
                 {NEXT SENTENCE         }
```

De te specificeren identifier is dezelfde als die waaraan de INDEXED BY-passage in de Data Division werd gekoppeld, bijvoorbeeld:

```
SET TAB-INDEX TO 1.
SEARCH TABEL-ELEMENT AT END PERFORM ELEMENT-NIET-GEVONDEN
    WHEN TABEL-ELEMENT(TAB-INDEX) = INVOER-CODE
    NEXT SENTENCE.
```

Een automatische binaire zoekmethode verkrijgen we door de SEARCH ALL-variant. In dit geval moet ook de rangorde van de tabel worden gespecificeerd, bijvoorbeeld:

```
02 TABEL-ELEMENT PICTURE XX OCCURS 100 TIMES
   ASCENDING KEY IS TABEL-ELEMENT
   INDEXED BY TAB-INDEX.
```

Om een binaire zoekmethode op gang te krijgen kunnen we zeggen:

```
SEARCH ALL TABEL-ELEMENT
    AT END PERFORM ELEMENT-NIET-GEVONDEN
    WHEN TABEL-ELEMENT(TAB-INDEX) = INVOER-CODE
    NEXT SENTENCE.
```

Er is geen opgave bij deze paragraaf. De volledige formaten van de opdrachten SEARCH en SET worden gegeven in Appendix D.

Het gebruik van INDEXED BY bij het werken met tabellen wordt aanbevolen, zelfs als er niet met SEARCH wordt gewerkt. De programmeur moet er dan echter goed op letten dat de waarde van de

index niet buiten het bereik van de tabelgrenzen valt. Dat is namelijk niet toegestaan, behalve wanneer de index het element is dat in de VARYING-passage van een PERFORM ... VARYING voorkomt.

O6 TWEEDIMENSIONALE TABELLEN

Deze worden gevonden wanneer een element wordt herhaald binnen een ander element dat zelf wordt herhaald, dat wil zeggen wanneer de ene OCCURS-passage ondergeschikt is aan een andere OCCURS-passage. Veronderstel bijvoorbeeld dat een fabrikant overzeese agenten, streekvertegenwoordigers en aannemers voorziet van onderdelen met een verschillend kortingspercentage overeenkomstig de aard van het onderdeel en de soort afnemer:

Afnemer	Onderdelen			
	Motoren	Panelen	Wielen	Andere
Overzeese agent	35%	30%	30%	30%
Streekvertegenw.	30%	25%	20%	20%
Aannemer	10%	5%	5%	0%

Veronderstel dat het programma een factuur moet klaarmaken vanuit een klantenorderrecord dat naast de bruto orderkosten de volgende twee code-elementen van ieder één cijfer bevat:

AFNEMERS-CODE

1 = overzeese agent
2 = streekvertegenw.
3 = aannemer

ONDERDEELSOORT

1 = motoren
2 = panelen
3 = wielen
4 = andere

De tabel met kortingspercentages kunnen we nu als volgt coderen:

```
WORKING-STORAGE SECTION.
01  KORTINGS-TABEL VALUE "353030303025202010050500".
    02 RIJ OCCURS 3 TIMES.
       03 KORTING OCCURS 4 TIMES PICTURE V99.
```

Dit programmadeel beschrijft een tabel met 3 rijen en 4 kolommen; de eerste vier elementen in de VALUE-passage zijn de kortingen voor iedere kolom van de eerste rij van de tabel, de volgende vier zijn de kortingen voor iedere kolom van de tweede rij, enz.

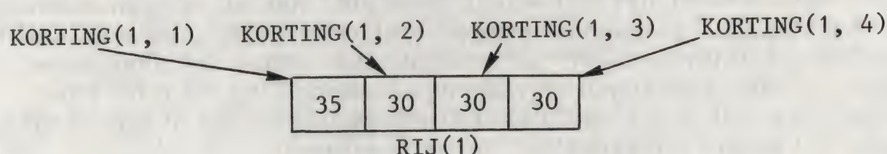
Een individueel element kan worden gelokaliseerd door twee indices, de eerste voor de rij en de tweede voor de kolom. Motoronderdelen voor overzeese agenten hebben de korting in de eerste rij, eerste kolom; dit is KORTING(1, 1). Panelen voor streekvertegen-

woordigers krijgen KORTING(2, 2), terwijl 'andere' voor aannemers KORTING(3, 4) oplevert. De inhoud van de VALUE-passage kan worden voorgesteld zoals in figuur 41:

RIJ(1)				RIJ(2)				RIJ(3)			
35	30	30	30	30	25	20	20	10	05	05	00

FIGUUR 41 Inhoud van KORTINGS-TABEL.

terwijl binnen een rij ieder element als volgt wordt beschreven:



FIGUUR 42 Inhoud van RIJ(1).

In dit voorbeeld beschrijft de waarde van AFNEMERS-CODE precies de gewenste rij, terwijl de waarde van ONDERDEELSOORT precies de gewenste kolom beschrijft; dat wil zeggen de tabel is zelfindicerend door deze elementen. Opdrachten in de Procedure Division kunnen bevatten:

```
MULTIPLY BRUTO-KOSTEN BY
        KORTING(AFNEMERS-CODE, ONDERDEELSOORT)
    GIVING KORTING-BEDRAG ROUNDED.
```

Opgaven

1. Veronderstel dat in ons voorbeeld AFNEMERS-CODE de waarden 0, 1, 2 aanneemt. Hoe zou u de laatste opdracht moeten wijzigen?

2. In hetzelfde programma als het voorbeeld in de tekst wordt een record als volgt beschreven:

```
01 REGEL.
   02 RIJ-UIT OCCURS 3 TIMES.
      03 FILLER PICTURE XX.
      03 KORTING-UIT PICTURE 0.99B OCCURS 4 TIMES.
```

   Er zijn twee elementen PICTURE 9 in working storage die wij RIJ-TELLER en KOL-TELLER noemen. Schrijf opdrachten die iedere KORTING naar het overeenkomende veld KORTING-UIT verplaatsen. (advies: Neem bijvoorbeeld de opdracht PERFORM RIJ-VULLEN VARYING RIJ-TELLER FROM 1 BY 1 UNTIL RIJ-TELLER = 4 op. In RIJ-VULLEN kan een andere PERFORM-opdracht staan die de individuele velden verplaatst.)

3. Een programma bevat een grote tabel die gebruikt wordt om er totalen in bij te houden:

```
01 TABEL-1.
   02 RIJ OCCURS 100 TIMES INDEXED BY RIJTELLER.
      03 TOTAAL OCCURS 5 TIMES PICTURE S9(4)V99 COMP
                INDEXED BY KOL-TELLER.
```

Elk totaal moet een beginwaarde krijgen van nul, maar dit kan niet worden bewerkstelligd door middel van een value-passage, aangezien de occurs-passage is gespecificeerd. Bovendien is voor TOTAAL COMP gespecificeerd hetgeen op veel machines inhoudt dat het overbrengen van nullen naar het groepsveld TABEL-1 *niet* tot gevolg heeft dat de individuele TOTAAL-velden ook nul zullen bevatten. Er zal dus code geschreven moeten worden om elk tabel-element op nul in te stellen. Schrijf de opdrachten die hierin voorzien.

O7 STRING

Hoewel ze op het eerste gezicht nogal ontzagwekkend lijken, maken de opdrachten STRING en UNSTRING in bepaalde gevallen een veel gemakkelijker verwerking van gegevens mogelijk. De complete formaten worden in Appendix D gegeven; ik zal proberen het gebruik van deze opdrachten toe te lichten en iets van hun mogelijkheden te illustreren. Maar kennis van deze opdrachten is voor de rest van deze tekst niet noodzakelijk.

De STRING-opdracht, ontmanteld tot wat louter wezenlijk is, heeft het formaat:

```
STRING {identifier-1} [identifier-2] ...DELIMITED BY {literal-3}
       {literal-1   } [literal-2   ]                 {SIZE     }
INTO identifier-3.
```

Het ideale gebruik van deze opdracht vindt plaats wanneer we een aantal velden willen bijeenbrengen om één lange reeks van tekens in een nieuw veld te maken. Een typerend voorbeeld treedt bijvoorbeeld op wanneer we een boodschap op een beeldscherm willen afbeelden als onderdeel van een dialoog met een terminal-gebruiker. Veronderstel bijvoorbeeld dat we de volgende records hebben:

```
01 BEELDSCHERM-REGEL          PICTURE X(60).
01 ARTIKEL-RECORD.
   02 ARTIKEL-NUMMER          PICTURE 9(4).
   02 VAK-NUMMER              PICTURE 99.
   02 ARTIKEL-BESCHRIJVING    PICTURE X(20).
```

Veronderstel verder dat de inhoud van de velden ARTIKEL-NUMMER, VAK-NUMMER en ARTIKEL-BESCHRIJVING respectievelijk 8076, 16 en "STALEN BUIS.   " is. Deze velden kunnen dan in BEELDSCHERM-REGEL worden opgemaakt door:

```
MOVE SPACES TO BEELDSCHERM-REGEL.
STRING "ARTIKEL: "
ARTIKEL-NUMMER
" BESCHRIJVING: "
ARTIKEL-BESCHRIJVING
" VAK: "
VAK-NUMMER
DELIMITED BY SIZE INTO BEELDSCHERM-REGEL.
```

De passage DELIMITED BY SIZE heeft eenvoudig tot gevolg dat de genoemde velden in hun totaliteit aan de reeks worden toegevoegd. Dat wil zeggen: na uitvoering van deze opdracht bevat BEELDSCHERM-REGEL:

```
ARTIKEL: 8076 BESCHRIJVING: STALEN BUIS._________VAK: 16
```

In tegenstelling tot MOVE vult STRING niet automatisch de rechterkant van het ontvangende veld met spaties op; vandaar de voorafgaande MOVE in het voorbeeld.

Laten we verder veronderstellen dat we de af te beelden regel willen doen verschijnen als een doorlopende zin door de literal " VAK: " zo te verplaatsen dat deze onmiddellijk na de laatste letter van de beschrijving volgt. Indien de beschrijving altijd werd gevolgd door een punt, kan dit bereikt worden door:

```
MOVE SPACES TO BEELDSCHERM-REGEL.
STRING "ARTIKEL: "
ARTIKEL-NUMMER
" BESCHRIJVING: "
DELIMITED BY SIZE
ARTIKEL-BESCHRIJVING DELIMITED BY "."
" VAK: "
VAK-NUMMER
DELIMITED BY SIZE INTO BEELDSCHERM-REGEL.
```

Het begrenzingsteken wordt zelf niet verplaatst. Zo is het resultaat van deze opdracht:

```
ARTIKEL: 8076 BESCHRIJVING: STALEN BUIS VAK: 16
```

Het begrenzingsteken kan ook een reeks tekens zijn. Indien geen begrenzingsteken wordt tegengekomen terwijl de move plaatsvindt, wordt het gehele betrokken veld toegevoegd aan de reeks.

Het is duidelijk dat, wanneer de moves niet DELIMITED BY SIZE zijn, het feitelijke aantal aan elkaar geregen tekens in het ontvan-

gende veld niet constant is. De programmeur kan het aantal aan elkaar geregen tekens vastleggen door gebruikmaking van de POINTER-mogelijkheid, die verwijst naar een (elementair) gegevenselement dat het nummer van de volgende beschikbare positie in het ontvangende veld moet bevatten, waarbij we van links tellen (dat wil zeggen het feitelijke aantal tekens dat is verplaatst naar het veld plus 1). Indien bijvoorbeeld een element VOLGEND-TEKEN PICTURE 999 USAGE COMP was beschreven, kunnen we coderen:

```
MOVE 1 TO VOLGEND-TEKEN.
MOVE SPACES TO BEELDSCHERM-REGEL.
STRING "ARTIKEL: "
ARTIKEL-NUMMER
" BESCHRIJVING: "
DELIMITED BY SIZE
ARTIKEL-BESCHRIJVING DELIMITED BY "."
" VAK: "
VAK-NUMMER
DELIMITED BY SIZE INTO BEELDSCHERM-REGEL POINTER VOLGEND-TEKEN.
```

Na uitvoering van deze opdracht heeft VOLGEND-TEKEN de waarde 48. Wanneer een pointer expliciet is gespecificeerd, worden de tekens in het ontvangende veld geplaatst, beginnend bij de positie die door het pointerveld is gespecificeerd; vandaar de noodzaak om een beginwaarde voor VOLGEND-TEKEN in het voorbeeld vast te leggen. De pointer-mogelijkheid is met name waardevol indien we aan een al eerder gevormde reeks nog meer wensen toe te voegen; om ons voorbeeld te vervolgen:

```
STRING " UITVERKOCHT" DELIMITED BY SIZE
       INTO BEELDSCHERM-REGEL POINTER VOLGEND-TEKEN.
```

zal – aangenomen dat de waarde van VOLGEND-TEKEN op geen enkele wijze is veranderd – het volgende resultaat opleveren:

```
ARTIKEL: 8076 BESCHRIJVING: STALEN BUIS VAK: 16 UITVERKOCHT
```

Tenslotte kan het voorkomen dat het ontvangende veld te klein is om de gewenste reeks te accepteren. Deze omstandigheid kan worden ontdekt door het sleutelwoord OVERFLOW (zie Appendix D).

O8 UNSTRING

Het doel van de UNSTRING-opdracht is een reeks tekens uit een veld van verzending te verdelen, als kortere reeksen, over ontvangende velden.

Veronderstel bijvoorbeeld dat we de volgende velden in de Data Division hebben:

```
    02 VOLGEND-TEKEN          PICTURE 999 COMP.
    02 AANTAL-REGELS          PICTURE 9 COMP.
01  INVOER-ADRES              PICTURE X(150).
01  UITVOER-ADRES.
    02 NAAM                   PICTURE X(40).
    02 ADR-REGEL-1            PICTURE X(40).
    02 ADR-REGEL-2            PICTURE X(40).
    02 ADR-REGEL-3            PICTURE X(40).
    02 ADR-REGEL-4            PICTURE X(40).
```

Het record INVOER-ADRES bevat een naam en 2, 3 of 4 adresregels, onderling gescheiden door een schuine streep. Geen enkele adres-regel is langer dan 40 tekens. We willen elke adres-regel in het corresponderende veld UITVOER-ADRES krijgen en bovendien willen we weten hoevel adres-regels er zijn. Dit kan op de volgende wijze door UNSTRING worden bereikt:

```
MOVE ZERO TO AANTAL-REGELS.
UNSTRING INVOER-ADRES DELIMITED BY "/" INTO
NAAM
ADR-REGEL-1
ADR-REGEL-2
ADR-REGEL-3
ADR-REGEL-4
TALLYING IN AANTAL-REGELS
ON OVERFLOW PERFORM TEVEEL-REGELS.
```

De OVERFLOW-conditie ontstaat indien alle ontvangende velden zijn gebruikt en er nog steeds niet in aanmerking genomen tekens in het veld van verzending over zijn. De TALLYING-mogelijkheid brengt een telling van het aantal ontvangende velden tot stand; merk op dat de programmeur het element waarin de telling wordt bijgehouden de gewenste beginwaarde moet geven. Gegevens naar de ontvangende velden worden gezonden volgens de normale regels voor MOVE (het veld van verzending wordt alfanumeriek behandeld), zodat er geen noodzaak is het ontvangende veld van een beginwaarde te voorzien.

Evenals bij STRING bestaat er een POINTER-mogelijkheid en meer dan één begrenzingsteken kan worden gespecificeerd door het bij elkaar voegen van begrenzingstekens met behulp van OR. Het begrenzingsteken zelf kan, indien gewenst, door de passage DELIMITED IN worden behouden. Laten we aannemen dat in het vorige voorbeeld het einde van de laatste adresregel werd gemarkeerd door een asterisk in plaats van de schuine streep; aannemende dat passende elementen zijn beschreven in de Data Division, kunnen we al deze mogelijkheden illustreren aan een eenvoudige routine die de naam en het adres afdrukt op achtereenvolgende regels:

```
P-ADRES-AFDRUKKEN.
    MOVE ZERO TO AANTAL-REGELS.
    MOVE 1 TO VOLGEND-TEKEN.
    MOVE SPACE TO SCHEIDINGSTEKEN.
    PERFORM P1-REGEL-AFBEELDEN UNTIL SCHEIDINGSTEKEN = "*".
    ADD AANTAL-REGELS TO TOTAAL-AANTAL-REGELS.
    :
    :
P1-REGEL-AFDRUKKEN.
    UNSTRING INVOER-ADRES DELIMITED BY "/" OR "/*"
        INTO BEELDSCHERM-REGEL
        DELIMITER IN SCHEIDINGSTEKEN
        POINTER VOLGEND-TEKEN
        TALLYING AANTAL-REGELS.
    WRITE BEELDSCHERM-REGEL BEFORE ADVANCING 1 LINE.
```

Een begrenzingsteken kan worden voorafgegaan door het woord ALL, in welk geval verschillende opeenvolgende begrenzingstekens worden behandeld als één gebeurtenis. In het vorige voorbeeld zou de routine, indien twee opeenvolgende schuine strepen in INVOER-ADRES worden ontmoet, een regel wit voor het tweede begrenzingsteken hebben uitgevoerd (indien er geen gegevens zijn die moeten worden verplaatst door de UNSTRING-opdracht, wordt het ontvangende veld gevuld met spaties als het alfanumeriek is, met nullen als het numeriek is). Dit kan worden ondervangen door:

```
UNSTRING INVOER-ADRES DELIMITED BY ALL "/" ...
```

Wanneer de programmeur wil weten hoeveel tekens zijn overgebracht naar een ontvangend veld, zal de COUNT IN-mogelijkheid in een teller voorzien (zie Appendix D).

ANTWOORDEN HOOFDSTUK O

Paragraaf O1

1. Stel u voor dat er een puntkomma staat in UITVOER-REC; om de andere machine dit als een puntkomma te laten lezen, moet u de puntkomma uitvoeren als een punt. Daarom moeten puntkomma's worden vertaald in punten. Wanneer we deze gedachtengang toepassen op ieder teken, krijgen we:

```
INSPECT UITVOER-REC
REPLACING ALL ";" BY "."
"." BY ";"
"*" BY "/"
"=" BY "*"
```

```
"/" BY "="
"#" BY ")"
"&" BY "("
")" BY "#"
"(" BY "&".
```

Paragraaf O2

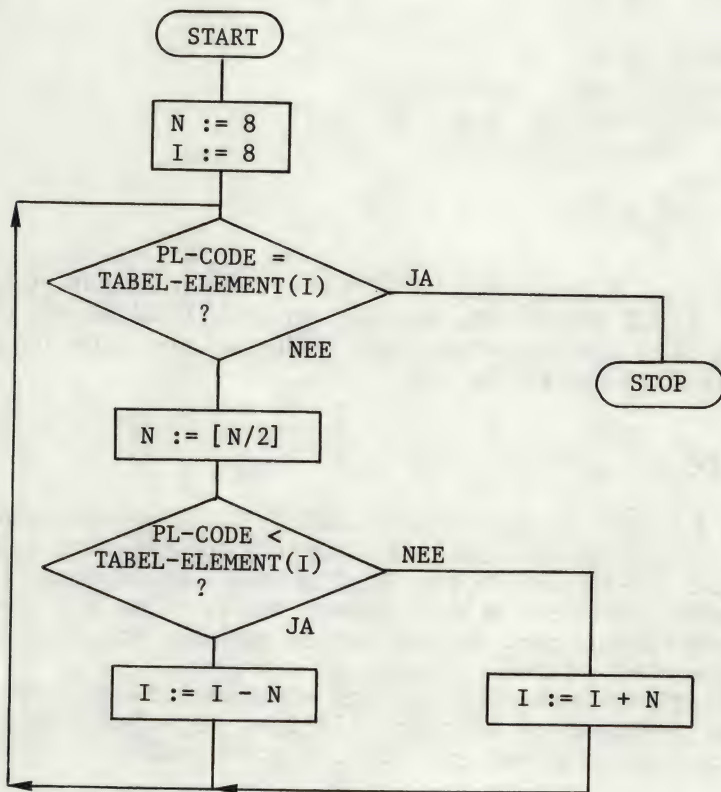
1. Van de vele mogelijke antwoorden volgt deze nauwgezet de tekst.

```
01  IN-REC.
    02 IN-TEKEN PICTURE X OCCURS 80 TIMES.
WORKING-STORAGE SECTION.
01  BESTANDS-INDICES COMP SYNC RIGHT.
    02 INREC-TELLER     PICTURE 999.
    02 NIEUWREC-TELLER  PICTURE 999.
01  TABEL.
    02 BRONTABEL VALUE "ABCDEFGHI123456789".
       03 BRONTEKEN PICTURE X OCCURS 18 TIMES.
    02 DOELTABEL VALUE "123456789ABCDEFGHI".
       03 DOELTEKEN PICTURE X OCCURS 18 TIMES.
    02 TABEL-TELLER PICTURE 99 COMP SYNC RIGHT.
01  NIEUW-REC.
    02 NIEUW-TEKEN PICTURE X OCCURS 240 TIMES.
PROCEDURE DIVISION.
:       (openen bestand en inlezen van KAART enz.)
:
    PERFORM B1-REGEL-VERTALEN UNTIL EVB-IN-BESTAND = "T".
        :
        :
B1-REGEL-VERTALEN.
    MOVE SPACES TO NIEUW-REC.
    MOVE 1 TO NIEUWREC-TELLER.
    PERFORM C1-TEKEN-VERTALEN VARYING
       INREC-TELLER FROM 1 BY 1 UNTIL INREC-TELLER = 81.
    READ IN-BESTAND AT END MOVE "T" TO EVB-IN-BESTAND.
C1-TEKEN-VERTALEN.
    IF IN-TEKEN(INREC-TELLER) = "*"
       PERFORM D1-MOVE-ZERO 3 TIMES
    ELSE
       PERFORM E1-VERGELIJKEN VARYING
          TABEL-TELLER FROM 1 BY 1 UNTIL
          TABEL-TELLER = 19 OR IN-TEKEN(INREC-TELLER) =
                               BRONTEKEN(TABEL-TELLER)
       IF TABEL-TELLER LESS THAN 19
          MOVE DOELTEKEN(TABEL-TELLER) TO NIEUW-TEKEN(NIEUWREC-TELLER)
          ADD 1 TO NIEUWREC-TELLER
       ELSE
          MOVE IN-TEKEN(INREC-TELLER) TO NIEUW-TEKEN(NIEUWREC-TELLER)
          ADD 1 TO NIEUWREC-TELLER.
```

```
D1-MOVE-ZERO.
    MOVE ZERO TO NIEUW-TEKEN(NIEUWREC-TELLER)
    ADD 1 TO NIEUWREC-TELLER.
E1-VERGELIJKEN.
```

Paragraaf O4

1. Laat de tabel 16 elementen bevatten, waarbij de laatste twee HIGH-VALUES zijn.



FIGUUR 43 Antwoord bij opgave 1, § O4.

```
WORKING-STORAGE SECTION.
01  TABEL.
    02 FILLER PICTURE X(70) VALUE "ASSENBEST DELFTEDE  EPE
-      "  HOORNKOOG LARENLISSEMOKUMOMMENSNEEKVAALSVENLO".
    02 FILLER PICTURE X(10) VALUE HIGH-VALUES.
```

```
01  TABEL-2 REDEFINES TABEL.
    02 PLAATS-NAAM OCCURS 16 TIMES.
       03 TABEL-ELEMENT  PICTURE XX.
       03 FILLER         PICTURE XXX.
01  HULPVELDEN COMP SYNC RIGHT.
    02 N  PICTURE 99.
    02 I  PICTURE 99.
PROCEDURE DIVISION.
    .
    .
    .
    MOVE 8 TO N
    MOVE 8 TO I.
    PERFORM B1-BINAIRE-ZOEK UNTIL PL-CODE = TABEL-ELEMENT(I).
    MOVE PLAATS-NAAM(I) TO enz.
    .
    .
    .
B1-BINAIRE-ZOEK.
    DIVIDE 2 INTO N ROUNDED.
    IF PL-CODE LESS THAN TABEL-ELEMENT(I)
       SUBTRACT N FROM I
    ELSE
       ADD N TO I.
```

Opmerking: In programma en stroomschema is er vanuit gegaan dat PL-CODE één van de waarden AS t/m VE uit de tabel bevat. Als we die veronderstelling laten vallen zal een extra controle in het programma nodig zijn.

Paragraaf O6

1. U zou 1 moeten optellen bij AFNEMERS-CODE om deze code in de reeks te brengen die gewenst is om hem als een index te gebruiken. Dit zou de waarde van AFNEMERS-CODE wijzigen, hetgeen ongewenst kan zijn, in welk geval u het resultaat zou moeten plaatsen in een apart element van de working storage en dat gebruiken als de index voor de rij KORTING.
   COBOL biedt de mogelijkheid van *relatieve* indicering om in dergelijke problemen te voorzien. Hier kan een index ieder element plus of min een geheel getal zijn, bijvoorbeeld:

   ```
   MULTIPLY BRUTO-KOSTEN BY KORTING(AFNEMERSCODE + 1,
       ONDERDEELSOORT) GIVING KORTING-BEDRAG ROUNDED.
   ```

2.
   ```
   PERFORM B1-RIJ-VULLEN VARYING RIJ-TELLER FROM 1 BY 1
       UNTIL RIJ-TELLER = 4.
   .
   .
   .
   ```

```
B1-RIJ-VULLEN.
    PERFORM C1-MOVE-KORTING VARYING KOL-TELLER FROM 1 BY 1
       UNTIL KOL-TELLER = 5
C1-MOVE-KORTING.
    MOVE KORTING(RIJ-TELLER, KOL-TELLER) TO
       KORTING-UIT(RIJ-TELLER, KOL-TELLER).
```

3.

```
    PERFORM B1-MET-NULLEN-VULLEN VARYING RIJ-TELLER
       FROM 1 BY 1 UNTIL RIJ-TELLER GREATER THAN 100.
    :
    :
B1-MET-NULLEN-VULLEN.
    PERFORM C1-TOTAAL-OP-NUL VARYING KOL-TELLER
       FROM 1 BY 1 UNTIL KOL-TELLER GREATER THAN 5.
C1-TOTAAL-OP-NUL.
    MOVE ZERO TO TOTAAL(RIJ-TELLER, KOL-TELLER).
```

P HET PROGRAMMEREN VAN INTERACTIEVE DIALOGEN

P1 ALGEMENE RICHTLIJNEN

Een dialoog kan eenvoudig te programmeren zijn, wanneer het om een reeks vragen en antwoorden gaat zonder veel vertakkende logica. Anderzijds kan het om een zeer gecompliceerd programma gaan, wanneer de gebruiker veel keuzemogelijkheden heeft en de dialoog zich langs vele verschillende wegen kan bewegen. Twee aspecten van interactieve programmering in de praktijk, die deze programmering bemoeilijken, zijn:
(a) het besturen van de beeldschermapparatuur, vooral wanneer deze beschikt over een groot aantal ingebouwde toepassingsmogelijkheden, en
(b) het besturen van programmalogica wanneer de keuzemogelijkheden gecompliceerd zijn.

Helaas is er geen standaardantwoord op (a). Beeldschermeenheden komen op de markt met een grote verscheidenheid aan toepassingsmogelijkheden, zoals omgekeerde video, vette letters en onderstreping, schermbeveiliging, X-Y cursor-besturing, schermverheldering, opflikkerende velden, geluidssignalen, etc. Deze toepassingsmogelijkheden worden gewoonlijk opgeroepen door de uitvoer naar het beeldscherm van besturingssymbolen die in het programma zijn opgeslagen, maar deze symbolen verschillen per apparaat. Verder bestaat er in de praktijk geen standaardaanpak voor het omgaan met beeldschermeenheden in COBOL (ofschoon CODASYL hier wel over denkt); bij de meeste methoden zijn eenvoudige teletype-achtige bewerkingen met READ en WRITE mogelijk. Sommige methoden gebruiken ACCEPT en DISPLAY voor in- en uitvoer (paragraaf Q3). Weer andere methoden behandelen het scherm als een record in een relatief bestand dat kan worden gelezen of waarheen kan worden geschreven (paragraaf T3). Nog weer andere methoden maken gebruik van de faciliteiten van de COMMUNICATIONS SECTION in standaard COBOL (niet in dit boek behandeld). Vele programmeringsafdelingen in het bedrijfsleven gebruiken voor de interactieve programmering gewijzigde COBOL-versies of hulpprogramma's voor een speciale schermopmaak en voor de gegevensinvoer. We adviseren u de technische handboeken te bestuderen van de apparatuur die u zelf gebruikt.

De volgende drie paragrafen betreffen specifieke methoden voor

de besturing van de logica in interactieve programma's. Ter wille van de eenvoud gaan we in paragraaf P2 en P3 uit van een teletype-achtig beeldscherm en in paragraaf P4 van een beeldscherm met X-Y cursorbesturing.

Van programmeurs wordt vaak verwacht dat ze een flink aandeel hebben in de analyse en het ontwerp van interactieve procedures. In de hoofdstukken 8 en 9 van mijn boek *Systems Analysis* worden ideeën aangedragen over het kiezen van passende dialogen en het ontwerpen van een schermopmaak. De volgende controlelijst kan ook van nut zijn:

1. Houd rekening met lettertypeveranderingen in antwoorden.
2. Organiseer grote menu's in een boomschema.
3. Denk aan het in drukvorm uitbrengen van de menu's.
4. Geef ervaren gebruikers de kans om af te korten of stukken af te snijden.
5. Denk aan verschillende graden van ondersteuning of verschil in lengte van de toelichtende boodschappen voor gebruikers met van elkaar verschillende ervaring.
6. Laat gebruikers het systeem zo mogelijk aan hun eigen zienswijze aanpassen.
7. Houd de dialoogvorm, werkmethode en betekenis van codes, sleutelwoorden, etc. door het gehele systeem heen standaard.
8. Geef snel antwoord op iedere invoer.
9. Zorg ervoor dat de gebruiker niet op zijn geheugen hoeft te vertrouwen.
10. Overweeg het produceren van afgedrukt materiaal in plaats van/naast een afbeelding op het scherm.
11. Houd gegevens bij van het systeem en gebruikersreacties met het oog op latere verbeteringen (bijvoorbeeld aantallen transacties, frequentie waarin verschillende keuzemogelijkheden werden uitgevoerd, frequentie van fouten).
12. Geef evenveel aandacht aan foutcondities als aan de hoofddialoog.
13. Kies uw woorden zorgvuldig.
14. Vermijd rommelige schermen.

P2 DE MENU-DIALOOG

Het menu is een gemakkelijk programmeerbare dialoog en is bijzonder geschikt voor onervaren gebruikers. Het grootste nadeel ervan is dat het menu langzaam kan zijn in vergelijking met andere dialoogvormen, vooral als de terminal langzaam werkt.

Menu's neigen naar een zekere wijdlopigheid, dus om de beknopte stijl van dit boek aan te houden moet ik een onrealistisch kort voorbeeld kiezen. Stel dat we het volgende hoofdmenu hebben:

Menu 1

Wilt u
1. een transactie invoeren
2. een bestand afdrukken
3. een vraag stellen?

Een antwoord '1' brengt het volgende menu:

Menu 1.1

Kies het type transactie:
1. factuur
2. betaling
3. mutatie op stambestand.

Een antwoord '2' op het hoofdmenu resulteert in deze afbeelding:

Menu 1.2

Kies af te drukken bestand:
1. facturen van deze maand
2. betalingen van deze maand
3. klantenstambestand

enz.

Laten we aannemen dat in dit systeem een blanco antwoord inhoudt dat de gebruiker niet met het menu wil doorgaan en moet worden teruggegeven naar het direct daarboven liggende menu. Een antwoord dat noch blanco noch een cijfer in het gebied van het menu is, is een foutief antwoord. Na het geslaagd verlopen van een bewerking op het laagste niveau (van een 'blad' in de boom van keuzemogelijkheden) moet het laatst aangeboden menu worden herhaald. Zo kan de gebruiker weer teruggaan in de boom met menu's door achtereenvolgens blanco antwoorden in te voeren, wanneer hem een ongewenst menu wordt aangeboden.

Deze hiërarchische vertakkende logica is duidelijk een 'case'-constructie. In wezen heeft het programma deze vorm:

Menu 1

```
Herhalen totdat blanco antwoord
    |Herhalen totdat antwoord geldig is
    |    |Beeld menu af
    |    |Zorg voor antwoord
    |    |Evalueer antwoord
    |    |Wanneer 1
    |    |     |Voer 'menu 1.1' uit
    |    |Wanneer 2
    |    |     |Voer menu 1.2' uit
    |    |Wanneer 3
    |    |     |Voer 'menu 1.3' uit
```

```
|     |Wanneer blanco
|     |     |geen handeling
|     |Anders
|     |     |antwoord niet geldig
```

Menu 1.1

```
Herhalen totdat blanco antwoord
     |Herhalen totdat antwoord geldig is
     |     |Beeld menu af
     |     |Zorg voor antwoord
     |     |Evalueer antwoord
     |     |Wanneer 1
     |     |     |Voer 'Factuur-invoeren' uit
     |     |Wanneer 2
     |     |     |Voer 'Betaling-invoeren' uit
     |     |Wanneer 3
     |     |     |Voer 'Stam-veranderen' uit
     |     |Wanneer blanco
     |     |     |geen handeling
     |     |Anders
     |     |     |antwoord is niet geldig
enz.
```

(De Evalueer ... Wanneer ... manier van omgaan met de case-constructie hier stemt overeen met een voorgestelde EVALUATE-instructie in COBOL.)

Het is duidelijk dat ieder antwoord waarvan niet specifiek is vastgesteld dat het niet geldig is, een geldig antwoord is. Ook dient te worden beseft dat instructies moeten worden toegevoegd aan de Anders-tak om de desbetreffende fout af te handelen. Deze schets veronderstelt dat het menu bij iedere foute invoer opnieuw wordt afgebeeld; willen we dit niet dan moet de handeling 'beeld menu af' verplaatst worden net vóór de binnenste herhalingsinstructie en de instructies die de fout afhandelen moeten het verplaatsen van de cursor naar het gegevensinvoerpunt ('zorg voor antwoord') na afbeelding van de foutmelding omvatten.

Verder moet worden beseft dat het woord 'antwoord' in dit voorbeeld *lokaal* is voor iedere procedure, dat wil zeggen antwoord in Menu 1 betekent het Menu 1-antwoord, terwijl antwoord in Menu 1.1 het Menu 1.1-antwoord betekent. De volgende algemene regels moeten in acht worden genomen bij het gebruik van gestructureerd Nederlands ten behoeve van het ontwerpen van programma's.

1. Verwijzingen naar bestandsnamen of gegevenselementen in centraal gedocumenteerde bestanden, bijvoorbeeld in een data dictionary, zijn globaal tenzij ze lokaal zijn gedeclareerd.
2. Verwijzingen naar andere gegevenselementen zijn lokaal tenzij globaal gedeclareerd.
3. Lokale elementen die moeten worden meegegeven als parameters bij de aanroep van een subroutine moeten tussen haakjes worden geplaatst na de vermelding van de naam van de subroutine.

Als het antwoord globaal was gedeclareerd in dit voorbeeld, zou dit tot gevolg hebben dat een blanco invoer waar dan ook de gebruiker direct naar de top van de boom zou voeren en uit het programma.

Bij de meeste computersystemen kan de tekst van de dialogen beter opgeslagen worden in voor het programma toegankelijk achtergrondgeheugen dan opgenomen als constanten in het programma. De reden hiervan is dat het gewoonlijk gemakkelijker is een gegevensbestand te redigeren dan een programma te wijzigen en opnieuw te compileren en omdat dialoogtekst tijdens het gebruik opeenvolgende aanpassingen en verfijningen zal ondergaan. Eén mogelijkheid is om ieder menu als een record in een bestand (direct-toegankelijk is waarschijnlijk het beste – zie hoofdstuk T) te hebben, terwijl een andere mogelijkheid is om ieder menu als een bestand te hebben en iedere keuze als een record in dat bestand. De laatste methode is flexibeler indien het besturingssysteem voldoende krachtig is voor het werken met de grote hoeveelheid bestanden die hieruit kan resulteren. Deze methode maakt het ook gemakkelijker een universeel-toepasbaar systeem voor het werken met menu's op te zetten.

Met een universeel-toepasbaar systeem voor het werken met menu's behoeft er slechts één programma te worden geschreven om de afbeelding van de menu's en de verkrijging van de antwoorden te regelen. Ieder menu-elementrecord moet de boodschap bevatten van dit element zelf die moet worden afgebeeld als het element wordt gekozen, en de te volgen handeling. Dit zal òf weergave van een ander menu (doordat de naam van een menubestand wordt opgegeven) zijn òf het uitvoeren van een 'blad'-handeling (doordat de naam van een programma wordt opgegeven) òf beide. Met deze strategie is het programmeren van een hiërarchische menu-dialoog eenvoudig herleid tot het maken van een verzameling gegevensbestanden voor de menu's en het schrijven van de bladprogramma's (de bladprogramma's kunnen worden uitgevoerd door middel van interprogrammatische communicatie – zie hoofdstuk U).

Opgave

1. Wijzig het voorbeeld van deze paragraaf zodanig, dat de dialoog terugkeert naar het hoofdmenu (Menu 1) indien de gebruiker de letter M intikt als antwoord op ieder verzoek om menukeuze.

Praktijk-opgave

1. Schrijf een specificatie in gestructureerd Nederlands van een universeel-toepasbaar systeem voor het werken met menu's. Werk het systeem uit en voer het in.
(Als u de dingen eenvoudig wilt houden, beperk dan de keuzemogelijkheden bij de menuselectie tot de cijfers 1,2,3, etc. en blanco. U zult het aantal keuzemogelijkheden, toegestaan bij een bepaald menu, moeten beperken.)

P3 STATUSOVERGANGSTABELLEN

Interactieve dialogen brengen vaak interpretatie van reeksen tekens met zich mee die zijn ingevoerd op het toetsenbord (dit soort problemen doet zich minder voor bij batch-systemen). Vele van deze problemen kunnen worden opgelost met de al toegelichte UNSTRING- en INSPECT-voorzieningen. Er is ook een tamelijk grote categorie problemen die kan worden opgelost door middel van statusovergangstabellen. Het belangrijkste voordeel van deze tabellen is dat ze helpen bij het verhelderen van een complex probleem vóór het coderen.

Stel dat in een interactief programma een terminalmedewerker een gegevenselement 'waarde verkoop' moet invoeren. Hoewel met 10 tekenposities voor het gegevenselement rekening moet worden gehouden, kan het gegeven iedere lengte tot en met 10 tekens hebben, inclusief een ongespecificeerd aantal spaties aan kop en staart. Het bedrag wordt niet van een rekenkundig teken (+ of -) voorzien. Minimaal één cijfer moet aanwezig zijn en een decimale komma kan facultatief aanwezig zijn. Een geheel getal moet worden geïnterpreteerd als zoveel 'gulden', tenzij het wordt gevolgd door het symbool c, in welk geval het moet worden geïnterpreteerd als 'cent'. Het symbool c is niet toegestaan, wanneer er een decimale komma aanwezig is.

Om een statusovergangstabel voor deze situatie op te zetten identificeren we alle tekens of categorieën tekens die van belang zijn voor de interpretatie van de gegevens - spaties, cijfer, decimale komma, symbool c, ieder ander teken. Dit zijn de tekens die een verandering van status bewerkstelligen, zoals later wordt beschreven. Alle cijfers worden samengenomen in de categorie 'cijfer', omdat er geen verschillende behandeling is vereist voor een bepaald cijfer. Tegelijkertijd zal ieder ander teken dan spatie, cijfer, decimale komma en cent-symbool aanleiding geven tot een foutstatus.

In dit geval bestaat er één andere van belang zijnde status van de invoer, namelijk de status waarbij het veld volledig vol is als gevolg van de invoer van tien tekens.

De invoerstatussituaties en tekens worden gebruikt om de kolommen van de tabel een naam te geven. Bij de constructie van de tabel geven we de eerste rij een benaming voor de oorspronkelijke status (bijvoorbeeld 'Vóór enig zinvol cijfer') en voeren we in de tabel onder iedere kolom een willekeurig getal in om de rij van de nieuwe status aan te geven. Natuurlijk gebruiken we hetzelfde getal, als twee verschillende invoerelementen leiden tot dezelfde nieuwe status. Geef de tweede rij een benaming en ga op deze wijze verder, totdat alle rijen afgehandeld zijn op de als afsluiting beschouwde statussituatie na.

Figuur 44 laat de hieruit voortvloeiende tabel zien.

Bij het maken van de tabel doen zich waarschijnlijk statussituaties voor die niet duidelijk waren voorzien in de verhalende specificaties (bijvoorbeeld kan de decimale komma het laatste teken ongelijk aan spatie zijn dat wordt ingevoerd?). Deze kwesties moeten ter

HUIDIGE STATUS \ INVOER	Spatie	Cijfer	Decimale komma	Symbool 'c'	Ieder ander teken	Einde van gegevens
1. Vóór enig zinvol cijfer	1	2	3	6	6	6
2. Cijfers voor decimale komma	5	2	3	5	6	7
3. Verwachting eerste cijfer na decimale komma	5	4	6	6	6	7
4. Verwachting tweede cijfer na decimale komma	5	5	6	6	6	7
5. Na einde gegevensinvoer	5	6	6	6	6	7
6. Fout						
7. Beëindigd						

FIGUUR 44 Statusovergangstabel met de nieuwe huidige status die voortvloeit uit de gespecificeerde invoertekens of de invoerstatus.

opheldering worden besproken met de analist of gebruiker.

In het voorbeeld zijn, om het kort te houden, statussituaties bij elkaar gezet die gedifferentieerd kunnen zijn als dat is vereist. Bijvoorbeeld: er is slechts één 'fout'-status; er kunnen echter verschillende soorten fouten onderscheiden zijn, zodat bijvoorbeeld als statussituaties voorkomen 'Fout-ongeldig teken aanwezig', of 'Fout-ingebedde spatie' of 'Fout-uitsluitend spatie als invoer', etc. (zie ook de volgende paragraaf). Evenzo hadden we statussituaties kunnen hebben als 'Beëindigd en het veld bevat gehele guldens', 'Beëindigd en het veld bevat gehele centen', etc.

Er bestaan directe mogelijkheden voor de implementatie van een programma vanuit een statustabel, maar wat de beste methode is, is afhankelijk van de omstandigheden. Een methode om nader te bekijken wordt hieronder beschreven. Deze lijkt een tikje gecompliceerd, maar als u de methode eenmaal begrijpt denk ik, dat u het niet op een andere manier wilt doen.

De kolommen worden geacht genummerd te zijn met 1,2,3,etc. en de statussituaties (= rijen) worden ook genummers met 1,2,3,etc. De statustabel wordt opgeslagen als gegevens en het invoerteken wordt omgezet, met behulp van een opzoektabel (zie paragrafen O2 en O5), in het kolomnummer. De procedure verloopt dan als volgt.

Valideer verkoopwaarde

```
Stel status in op 1
Voer-in (of indiceer) eerste teken
Herhalen totdat status = 6 of 7
    |Zet invoerteken om in kolomnummer
    |Geef status de waarde van tabelinhoud (rij = status,
                kolom = kolomnummer)
    |Als status niet = 6 of 7
    |    |Voer-in (of indiceer) volgende teken
```

Deze methode leidt tot redelijk gemakkelijk onderhoud, mits de programmeur die het onderhoud verzorgt de methode ook begrijpt en de statusovergangstabel is gedocumenteerd.

Opgave

1. Welke wijziging zou u moeten aanbrengen in figuur 44 als de decimale komma niet het laatste teken ongelijk aan de spatie in de invoer mocht zijn?

P4 STATUSTABEL TER BESTURING VAN DIALOOGVOORTGANG

Vaak doen zich situaties voor in interactieve dialogen waarbij het gewenst is de besturing te regelen op een niet-hiërarchische wijze. Bekijk bijvoorbeeld de volgende mogelijke dienstverlening aan een gebruiker die een aantal investeringsalternatieven wil overwegen.

```
Beantwoord ieder van de volgende vier vragen.
Antwoord met een ? voor het element dat u wilt vinden; voer
de waarde in van de andere drie elementen. Er is slechts
één ? toegestaan in de antwoorden.

Aantal investeringsjaren (= gehele jaren) = ...
Beginkapitaal = $ .....
Jaarlijks rentepercentage (samengesteld op jaarbasis) = ..... %
Eindkapitaal = $ .....

Druk RETURN in na iedere invoer.
Druk RETURN in als antwoord op welke vraag dan ook als u
een vergissing hebt gemaakt en opnieuw wilt beginnen bij
vraag 1.

zzzzzzzzzzzzzzzzzzzzzzzzzzzzz
```

Het gehele scherm moet bij aanvang worden gepresenteerd (behalve de regel met z's, die de plaats van het resultaat en het gebied van de foutboodschappen laat zien). De cursor wordt bewogen naar de eerste punt van de eerste vraag. Na geldige invoer van de gebruiker wordt de cursor geplaatst bij de eerste punt van de volgende vraag, etc. Na de laatste gegevensinvoer wordt een boodschap geschreven, die één van deze vier is in overeenstemming met het '?':

Aantal vereiste jaren is ZZ9
Vereiste beginkapitaal is $$,$$$,$$9
Benodigd rentepercentage is ZZ9,999%
Eindkapitaal bedraagt $$,$$$,$$9

Als zich een size error voordoet, luidt de boodschap:

Het antwoord is te groot voor weergave.
Voer nieuwe gegevens in.

Als de gegevens van de gebruiker niet geldig zijn, wordt een passende foutmelding afgebeeld in boodschapregel en wordt de cursor weer teruggeplaatst bij de eerste punt van het desbetreffende invoerelement. Als het invoerelement van de gebruiker leeg is (aangegeven door de invoer van alleen RETURN in deze specifieke situatie), wordt het scherm opnieuw gepresenteerd en de cursor verplaatst naar vraag 1.

Om dit probleem zo te herordenen dat we het aankunnen met een hiërarchische oplossing is bijzonder alledaags. In ons voorbeeld zou het één menu vereisen dat vier keuzemogelijkheden biedt. Voor iedere keuze zou er een andere dialoog zijn waarin wordt verzocht om de drie waarden benodigd om die specifieke vraag te beantwoorden.

De oplossing met behulp van de statustabel is in principe gelijk aan de in de vorige paragraaf beschreven oplossing. Het enige verschil is dat behalve een nieuwe status die bepaald wordt door de inhoud van een tabelelement, iedere combinatie van huidige status/invoer ook een *handeling* vereist die moet worden uitgevoerd alvorens verder te gaan met de volgende status.

Figuur 45 laat een statustabel voor ons probleem zien. Naast de nieuwe status is ieder element uit de tabel ook voorzien van een handelingsgetal. Dit is alleen maar gedaan, opdat u de implementatie van de statustabel begrijpt; het handelingsgetal wordt niet in de tabel maar in het programma opgenomen.

Net zo als daarnet is het noodzakelijk de tabel methodisch door te werken en de overgangen van de huidige naar de volgende status te analyseren, maar daarnaast de handeling te noteren die bij iedere fase moet worden uitgevoerd. Ter wille van de beknoptheid is de lijst met handelingen weggelaten, maar wel opgenomen in de in gestructureerd Nederlands gestelde programma-specificatie die onder figuur 45 volgt.

HUIDIGE STATUS / INVOER SITUATIE	1 "?"	2 geldig numeriek antwoord op vraag	3 iedere andere situatie	4 leeg
1. Voor vraag aantal jaren	A1, 2	A2, 3	A3, 1	A4, 1
2. ?gevonden, voor beginkapitaal vraag	A5, 2	A6, 4	A7, 2	A8, 1
3. ?niet gevonden, voor beginkapitaal vraag	A9, 4	A10,5	A11,3	A12,1
4. ?gevonden, voor percentage-vraag	A13,4	A14,6	A15,4	A16,1
5. ?niet gevonden, voor percentage-vraag	A17,6	A18,1	A19,5	A20,1
6. ?gevonden, voor eindkapitaal vraag	A21,6	A22,1	A23,6	A24,1

FIGUUR 45 Statusovergangstabel voor de investeringsdialoog.

Investeringsvraagprogramma

```
Herhalen totdat geen-vragen-meer
    |Beeld tekst af op scherm
    |Cursor naar veld aantal jaren
    |Stel status in op 1
    |Haal invoersituatie op
    |Herhalen totdat vraag-beantwoord
    |    |Doe 'Zet invoersituatie om in kolom-nr'
    |    |Doe 'Voer handeling uit (status, kolom-nr, vraag-
    |    |                        beantwoord)'
    |    |Exit als vraag-beantwoord
    |    |Geef status waarde van tabelinhoud (status, kolom-nr)
    |    |Haal invoersituatie op
    |Beeld af "Hebt u een andere vraag? Antwoord J of N"
    |Als antwoord is "N"
    |    |Geen-vragen-meer
```

Zet invoersituatie om in kolom-nr

```
Evalueer antwoord van gebruiker
Wanneer "?"
    |Stel kolom-nr in op 1
Wanneer spatie
    |Stel kolom-nr in op 4
Wanneer geldig numeriek antwoord op vraag
    |Stel kolom-nr in op 2
Anders
    |Stel kolom-nr in op 3
```

Voer handeling uit (status, kolom-nr, vraag-beantwoord)

```
Evalueer status en kolom-nr
Wanneer 1 en 1
     |Stel vraagtype in op 1
     |Cursor naar veld beginkapitaal
Wanneer 1 en 2
     |Cursor naar veld beginkapitaal
Wanneer 1 en 3
     |Schrijf boodschap dat gegevens ongeldig zijn
     |Cursor naar veld aantal-jaren
Wanneer 2 en 1
     |Schrijf boodschap dat ? al ingevoerd is
     |Cursor naar veld beginkapitaal
Wanneer 2 en 2
     |Cursor naar veld percentage
Wanneer (2 en 3) of (3 en 3)
     |Schrijf boodschap dat gegevens ongeldig zijn
     |Cursor naar veld beginkapitaal
Wanneer 3 en 1
     |Stel vraagtype in op 2
     |Cursor naar veld percentage
Wanneer 3 en 2
     |Cursor naar veld percentage
Wanneer 4 en 1
     |Schrijf boodschap dat ? al ingevoerd is
     |Cursor naar veld percentage
Wanneer 4 en 2
     |Cursor naar veld eindkapitaal
Wanneer (4 en 3) of (5 en 3)
     |Schrijf boodschap dat gegevens ongeldig zijn
     |Cursor naar veld percentage
Wanneer 5 en 1
     |Stel vraagtype in op 3
     |Cursor naar veld eindkapitaal
Wanneer 5 en 2
     |Bereken eindkapitaal
     |Als size error
     |    |Schrijf boodschap resultaat te groot
     |Anders
     |    |Schrijf boodschap eindkapitaal
     |Vraag-beantwoord
Wanneer 6 en 1
     |Schrijf boodschap dat ? al ingevoerd is
     |Cursor naar veld eindkapitaal
Wanneer 6 en 2
     |Evalueer vraagtype
     |Wanneer 1
     |    |Bereken jaren
```

```
|      |Als size error
|      |      |Schrijf boodschap resultaat te groot
|      |Anders
|      |      |Schrijf boodschap jaren
|Wanneer 2
|      |Bereken beginkapitaal
|      |Als size error
|      |      |Schrijf boodschap resultaat te groot
|      |Anders
|      |      |Schrijf boodschap beginkapitaal
|Wanneer 3
|      |Bereken percentage
|      |Als size error
|      |      |Schrijf boodschap resultaat te groot
|      |Anders
|      |      |Schrijf boodschap percentage
|Anders
|      |Fout programmalogica
|Vraag-beantwoord
Wanneer 6 en 3
     |Schrijf boodschap dat gegevens ongeldig zijn
     |Cursor naar veld eindkapitaal
Anders
     |Cursor naar veld beginkapitaal
```

Bij implementatie van de case-constructie door middel van GO TO ... DEPENDING ON HANDELINGS-NR kan het HANDELINGS-NR in deze situatie worden berekend door: (status-1)*4+kolom-nr.

Statusovergangstabellen kunnen worden gebruikt om dialogen beter te besturen met name bij de besturing van opeenvolgende schermpresentaties. In verschillende fasen en op verschillende analytische niveaus van een gecompliceerde dialoog zal men de besturing beschrijven met behulp van statustabellen, menu's en gestructureerde constructies. De programmeur die al deze drie technieken beheerst verkeert in de positie ieder commercieel op tekst gebaseerd dialoogprobleem op te lossen.

Praktijk-opgave

1 Implementeer het investeringsvraagprogramma op uw apparatuur.

ANTWOORDEN HOOFDSTUK P

Paragraaf P2

1. Declareer invoer-is-M globaal in Menu 1. Verander de openingsinstructie van ieder ondergeschikt menu in: Herhalen totdat blanco antwoord of invoer-is-M. In ieder ondergeschikt menu, geen handeling uitvoeren in het geval antwoord = "M".

Paragraaf P3

1. Verander de 5 in rij 3, kolom 1 in een 6.

DEEL 3

Voortgezet COBOL

Q HULPMIDDELLEN VOOR DE PROGRAMMEUR EN DIVERSEN

Q1 COPY

Het formaat van deze opdracht is eenvoudig:

```
COPY text-name
```

waarin 'text-name' de naam is van eerder geschreven COBOL-opdrachten die zijn opgeslagen in een voor de COBOL-compiler toegankelijke bibliotheek. Deze bibliotheek is in het algemeen vastgelegd op magneetschijf.

Veronderstel bijvoorbeeld dat we ervoor moeten zorgen dat de volgende codering wordt opgenomen in de bibliotheek onder de naam STANDAARD-KAARTLIJST:

```
SELECT KAART-IN ASSIGN TO CARD-READER.
SELECT AFDRUK ASSIGN TO PRINTER.
```

Als we een programma willen schrijven dat deze bestandsnamen en apparatuur gebruikt, kunnen we de paragraaf FILE-CONTROL van het programma schrijven als volgt:

```
FILE-CONTROL.  COPY STANDAARD-KAARTLIJST.
```

De compiler zal de opdrachten in de bibliotheek opzoeken en deze zo opnemen dat ons programma zal worden gecompileerd alsof we volledig hadden geschreven:

```
FILE-CONTROL.  SELECT KAART-IN ASSIGN TO CARD-READER.
               SELECT AFDRUK ASSIGN TO PRINTER.
```

COBOL stelt ons in staat een naam of woord in de bibliotheek-opdrachten te wijzigen door gebruikmaking van REPLACING, bijvoorbeeld:

```
FILE-CONTROL.  COPY STANDAARD-KAARTLIJST
               REPLACING KAART-IN BY KAART-BESTAND,
                         AFDRUK BY AFDRUK-BESTAND.
```

Dit wordt gecompileerd als:

```
FILE-CONTROL.  SELECT KAART-BESTAND ASSIGN TO CARD-READER.
               SELECT AFDRUK-BESTAND ASSIGN TO PRINTER.
```

COBOL stelt ons bovendien in staat de opdrachten te wijzigen of aan te passen, bijvoorbeeld COPY STANDAARD-KAARTLIJST REPLACING <<PRINTER>> BY <<PRINTER RESERVE 3 AREAS>> levert op:

```
FILE-CONTROL.  SELECT KAART-IN ASSIGN TO CARD-READER.
               SELECT AFDRUK ASSIGN TO PRINTER
                      RESERVE 3 AREAS.
```

Resteert nog de kwestie hoe we de standaardopdrachten in de bibliotheek krijgen. Dit wordt niet op een standaardmanier door de COBOL-compiler gedaan; computerfabrikanten verschaffen een speciaal programma (een *functieprogramma*; Engels: *utility*) dat opdrachten in de bibliotheek opneemt.

In een afdeling die een centrale catalogus of woordenlijst (dictionary) bijhoudt van alle gegevenselementen in de bestanden, kan het een goede gedachte zijn de bestands- en recordbeschrijvingen in de bibliotheek vast te leggen zodat deze kunnen worden gekopieerd in programma's. Afgezien van deze mogelijkheid heeft COPY een tamelijk beperkt gebruik in de praktijk. Maar het principe van de bibliotheek is van belang voor ander programmeringswerk.

Q2 HULPMIDDELLEN BIJ DEBUGGING (= HET OPSPOREN VAN FOUTEN)

Er bestaan diverse faciliteiten voor het opsporen van fouten. Voordat deze faciliteiten werden gestandaardiseerd, verschaften veel computerfabrikanten debugging-hulpmiddelen van eigen makelij (ze doen dat trouwens nog steeds). Er bestaan drie typen faciliteiten: debug-regels, trace-faciliteiten en monitoring-faciliteiten.

Debug-regels zijn opdrachten in een programma die tijdens het testen wel kunnen worden uitgevoerd, maar die niet zijn bedoeld als deel van het uiteindelijke programma. Iedere regel met een 'D' in kolom 7 is een debug-regel die in wezen als commentaar wordt behandeld door de compiler, tenzij hij bij de compilatie of uitvoering door de programmeur wordt geactiveerd.

Bij de compilatie kan de compiler worden gevraagd de debug-regels te compileren door het toevoegen van de uitdrukking WITH DEBUGGING MODE aan de paragraaf SOURCE-COMPUTER. Bij afwezigheid van deze uitdrukking worden debug-regels behandeld als commentaar.

Bij de uitvoering kan de operator maatregelen treffen waardoor de computer de gecompileerde debug-regels al of niet, zoals wordt verlangd, uitvoert. Dit vindt plaats door opdrachten aan en van het besturingssysteem.

Trace-faciliteiten beogen de paragraafnamen of regelnummers uit te lijsten in de volgorde waarin ze worden uitgevoerd door het programma.

Monitoring-faciliteiten beogen te laten zien hoe de waarde van een specifiek element verandert naarmate het programma vordert.

Debug-regels moeten in programma's worden opgenomen om de correctheid van veronderstellingen die tijdens het schrijven van het uitgangstaalprogramma werden aangenomen te toetsen. Afgezien van deze gebruiksmogelijkheid behoeven programmeurs die het gestructureerd ontwerpen van programma's beheersen, zelden een beroep te doen op deze voorzieningen.

Q3 MEDEDELINGEN VAN EN AAN DE OPERATOR

Een boodschap kan aan de operator worden overgebracht gedurende de uitvoering van het programma door middel van de opdracht DISPLAY. Deze opdracht bestaat eenvoudig uit het woord DISPLAY gevolgd door een reeks variabelen en/of literals, bijvoorbeeld

```
DISPLAY "AANTAL RECORDS GELEZEN = " REC-GETELD.
```

Als REC-GETELD 01273 bevat, zou de boodschap

```
AANTAL RECORDS GELEZEN = 01273
```

op het weergavemiddel verschijnen, dat wil zeggen de randapparatuur die ontworpen is voor visuele weergave. Gewoonlijk is dit het bedieningspaneel van de operator of het beeldscherm van de programmeur of gebruiker. Bij sommige computers kan het weergavemiddel nadrukkelijk vermeld worden.

Om informatie van de operator te verkrijgen wordt gebruik gemaakt van de opdracht ACCEPT. Het woord ACCEPT wordt gevolgd door een enkele variabele naam. De machine stopt en wat de operator nu ook typt wordt de inhoud van de variabele, op mogelijke afkapping na. Bijvoorbeeld

```
DISPLAY "VOER EEN GETAL VAN 2 CIJFERS IN"
ACCEPT AANTAL.
```

Wederom wordt bij sommige computers het invoerapparaat nadrukkelijk vermeld. Wat de operator intoetst wordt links-aansluitend naar het ontvangende veld overgebracht. Bij numerieke velden loont het dus de moeite de operator te laten weten uit hoeveel posities zo'n veld bestaat.

Er bestaat een tweede formaat voor de ACCEPT-opdracht:

```
                              DATE
ACCEPT identifier FROM      { DAY  }
                              TIME
```

De DATE-mogelijkheid plaatst een zescijferige datum in de vorm JJMMDD in de gespecificeerde identifier. DAY geeft een vijfcijferige Juliaanse datum, JJDDD. TIME geeft een achtcijferige tijd in uren (24-uurs kloksysteem), minuten, seconden en honderdsten van een seconde.

Q4 CONDITIENAMEN

Een handig hulpmiddel in verband met documentatie is de conditienaam op niveau-88. Hierbij wordt een naam (conditienaam) aan bepaalde *waarden* van gegevens in een elementair veld toegewezen. Als tijdens de uitvoering het element de vastgelegde waarde bevat, is de conditienaam *true* (waar); anders is hij *false* (niet waar). De waarheid van de conditienaam kan worden getest in een IF-opdracht in de Procedure Division. Bijvoorbeeld:

```
      02 FOUT-TEKEN PICTURE X.
         88 GELDIG-RECORD VALUE IS "F".
         88 ONGELDIG-RECORD VALUE IS "T".
         .
         .
         .
  PROCEDURE DIVISION.
         .
         .
         .
      PERFORM B1-VALIDEER-REC.
      IF ONGELDIG-RECORD PERFORM B2-FOUTMELDING.
```

De laatste zin heeft hetzelfde resultaat als

```
IF FOUT-TEKEN = "T" PERFORM B2-FOUTMELDING.
```

Conditienamen kunnen met ieder elementair veld worden verbonden. Een reeks waarden kan door de THRU-mogelijkheid worden aangegeven:

```
02 BESTELDE-HOEVEELHEID PICTURE 9(4).
   88 KLEINE-HOEVEELHEID VALUE IS 0 THRU 100.
   88 GROTE-HOEVEELHEID VALUE IS 101 THRU 9999.
```

De reeksen met waarden mogen elkaar overlappen indien gewenst. (Een goed voorstel van CODASYL is, dat een zin als SET GELDIG-RECORD TRUE hetzelfde effect moet hebben als MOVE "F" TO FOUT-TEKEN.)

Q5 STOP

Ik heb hiervoor nog geen moeite gedaan deze opdracht toe te lichten, omdat het doel ervan evident is.

Er bestaan twee versies:

```
STOP literal.
STOP RUN.
```

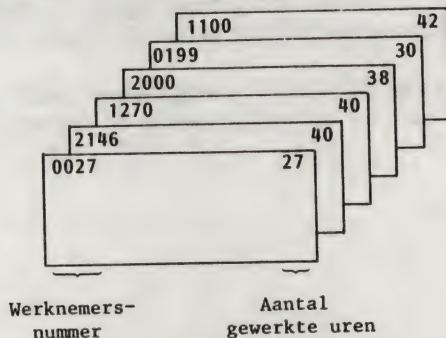
In het eerste geval stopt de machine met het uitvoeren van het programma en de literal wordt zichtbaar gemaakt aan de operator. De operator kan het programma opnieuw starten vanaf het punt waar werd gestopt.

In het tweede geval wordt het programma volledig beëindigd en wordt de besturing van de computer weer overgedragen aan het operating system.

R SORTEREN

R1 SORTERINGSBEGRIPPEN

Het doel van een sorteerbewerking bestaat in het plaatsen van de records van een bestand in een bepaalde volgorde. Veronderstel dat we een kaartenbestand zouden hebben als volgt:



FIGUUR 46 Bestand van TIJD-KAARTEN.

en we willen dit stellen tegenover een stambestand op magneetband waarop werknemersrecords staan in volgorde van werknemersnummer (één record per werknemer). Dan is het duidelijk dat het bestand van tijdkaarten in dezelfde volgorde moet worden gerangschikt als het stambestand.

In dit voorbeeld kunnen we dit gemakkelijk met de hand bereiken. Maar als het bestand groot is, of als het op magneetband is vastgelegd, moeten we de computer het sorteren voor ons laten doen. Hoe de computer deze sorteerbewerking voltooit is tamelijk ingewikkeld en gaat buiten het bestek van dit boek – het is voldoende voor ons te begrijpen dat er een programma bestaat, door de computerfabrikant geleverd, dat een niet-gesorteerd bestand inleest en een gesorteerd bestand oplevert in overeenstemming met onze instructies.

Het werknemersnummer in het voorbeeld wordt het *sleutelveld* van de sorteerbewerking genoemd, dat wil zeggen het is het veld

dat het gegeven bevat dat bepaalt waar in het bestand het record moet worden geplaatst. Het bestand kan zijn gesorteerd in *opklimmende* volgorde (het laagst genummerde record vooraan in het bestand) of *afdalende* volgorde (het hoogste nummer vooraan in het bestand).

Het sorteren van TIJD-KAARTEN op een computer gaat globaal als volgt. De kaarten worden ingelezen (de *eerste gang*) en overgebracht naar het achtergrondgeheugen om een *werkbestand* te vormen. De records in het werkbestand worden in het achtergrondgeheugen door elkaar geschud totdat ze zijn gerangschikt en tenslotte uitgevoerd (de *laatste gang*) op een eenheid die we hebben gespecificeerd, bijvoorbeeld magneetschijf.

Tijdens de eerste gang worden de records door de computer ingelezen in hun oorspronkelijke volgorde. Tijdens de laatste gang worden ze uitgevoerd in sleutelvolgorde met als resultaat het gesorteerde bestand.

Opgave

1. Als TIJD-KAARTEN in afdalende volgorde wordt gesorteerd met 'aantal gewerkte uren' als sleutelveld, welk record verschijnt dan als eerste in het bestand?

R2 PRIMAIRE EN SECUNDAIRE SLEUTELS

Soms is het wenselijk een bestand op meer dan één sleutelveld te sorteren. Veronderstel dat de bedrijfsleiding een verslag wil hebben van het aantal gewerkte uren in afdalende volgorde van het aantal uren. Er kunnen meerdere werknemers zijn die hetzelfde aantal uren hebben gewerkt, en in het verslag moeten deze werknemers in opklimmende volgorde van werknemersnummer staan.

Het aantal gewerkte uren is het *primaire sleutelveld* (het belangrijkste) en het werknemersnummer is het *secundaire sleutelveld* (het minst belangrijke). We moeten nu TIJD-KAARTEN sorteren in opklimmende volgorde van de secundaire sleutel binnen een afdalende volgorde van de primaire sleutel. Dit houdt in dat na sortering het eerste record van het bestand het record is van de werknemer die het meeste aantal uren heeft gewerkt; als er meer dan één werknemer is die ditzelfde aantal uren heeft gewerkt, staat de werknemer met het lagere werknemersnummer vooraan. Het laatste record in het gesorteerde bestand is het record van de werknemer die het minste aantal uren heeft gewerkt; zijn er meer met dit aantal, dan heeft dit laatste record het hoogste werknemersnummer ervan.

Sleutelvelden met numerieke pictures worden algebraïsch met elkaar vergeleken. Alle andere velden worden teken-voor-teken met

elkaar vergeleken overeenkomstig de *sorteervolgorde* van de specifieke computerfabrikant.

Een ASCII-sorteervolgorde kan worden opgelegd door de declaratie:

```
PROGRAM COLLATING SEQUENCE IS STANDARD
```

in de paragraaf OBJECT-COMPUTER van de Configuration Section; anders zal de eigen sorteervolgorde van de machine worden gebruikt (deze variëren). Er bestaan voorzieningen om niet-gestandaardiseerde sorteervolgordes op te roepen: raadpleeg het handboek.

Bij het sorteren van alfanumerieke velden moet ervoor worden gezorgd dat de verkregen volgorde ook de gewenste is. Er kunnen een aantal vreemde effecten optreden. Bij sommige sorteervolgordes bijvoorbeeld kan een rangschikking op een alfanumeriek veld, dat bij de invoer van de gegevens als een numeriek veld met teken was gedeclareerd, tot gevolg hebben dat de negatieve getallen hoger worden gesorteerd dan de positieve getallen.

R3 DE OPDRACHT SORT

In zijn eenvoudigste vorm wordt de opdracht SORT gebruikt om een invoerbestand te sorteren en een gesorteerd uitvoerbestand voort te brengen. Beide bestanden moeten worden gedefinieerd door conventionele FD-beschrijvingen.

Een extra werkbestand moet worden gedefinieerd met een SD-beschrijving (Sort-file Description). Dit is zuiver een hulpbestand dat wordt gebruikt om de sorteerbewerking te voltooien. De SD-beschrijving wordt gevolgd door record(01)-beschrijvingen. De bestandsnaam gespecificeerd in een SD-beschrijving moet met SELECT en ASSIGN worden behandeld zoals ieder ander bestand, maar sommige fabrikanten stellen speciale eisen aan de naam van de eenheid waaraan een sorteringsbestand wordt gekoppeld. In het gebruik kan de opdracht als volgt eruit zien:

```
FILE SECTION.
SD  WERKBESTAND.
01  GEWERKTE-TIJD.
    02 WERKNEMERS-NR   PICTURE X(4).
    02 FILLER          PICTURE X(74).
    02 UREN            PICTURE 99.
FD  TIJD-KAARTEN.
01  KAART-IN PICTURE X(80).
FD  GESORTEERDE-TIJD-KAARTEN.
01  RECORD-UIT PICTURE X(80).
```

```
PROCEDURE DIVISION.
A1.
    SORT WERKBESTAND ON
         DESCENDING UREN
         ASCENDING WERKNEMERS-NR
         USING TIJD-KAARTEN
         GIVING GESORTEERDE-TIJD-KAARTEN.
    STOP RUN.
```

Dit programma doet niets anders dan het sorteren van het bestand TIJD-KAARTEN, zodat het gesorteerde bestand (GESORTEERDE-TIJD-KAARTEN) in een volgend programma kan worden gebruikt, bijvoorbeeld het programma dat het verslag voor de bedrijfsleiding produceert. De primaire sleutel wordt het eerst, de secundaire sleutel het laatst gespecificeerd, dat wil zeggen in afnemende belangrijkheid (dit behoeft geen betrekking te hebben op de volgorde waarin ze verschijnen in het record). De sleutels moeten velden zijn in het sorteringsbestandrecord. Het is alsof KAART-IN wordt verplaatst naar GEWERKTE-TIJD om te worden gesorteerd en vervolgens verplaatst naar RECORD-UIT na het sorteren. De sleutelwoorden USING en GIVING specificeren respectievelijk de namen van het ongesorteerde invoerbestand en het gesorteerde uitvoerbestand.

Merk op dat het sorteringsbestand zelf niet wordt geopend. Dit wordt automatisch gedaan. In de praktijk verschilt het per computerfabrikant of de andere bestanden nu wel of niet moeten worden geopend en/of gesloten (in één systeem behoeven ze niet te worden geopend, maar is het wel vereist dat het uitvoerbestand wordt vergrendeld (closed with lock) als het bewaard moet blijven nadat het programma is voltooid).

Wanneer het management een overzicht had willen ontvangen met zowel het aantal gewerkte uren als het werknemersnummer in opklimmende volgorde, hadden we het gesorteerde bestand als volgt kunnen klaarmaken:

```
SORT WERKBESTAND
ON ASCENDING KEY UREN WERKNEMERS-NR
USING TIJD-KAARTEN
GIVING GESORTEERDE-TIJD-KAARTEN
```

Opgaven

1. Een magneetschijf bevat ongeordende records van 100 tekens voor boeken in een bibliotheek. De auteursnaam staat in de eerste 20 tekens, een classificatie in de volgende 10 tekens en de titel in de resterende tekens. Geef beschrijvingen van de DATA DIVISION en de PROCEDURE DIVISION om dit bestand te sorteren naar titel binnen auteur binnen classificatie (opklimmend).

2. Zal het hieruit resulterende bestand in de volgorde staan die de bibliothecaris verwacht?

R4 EIGEN CODERING – EERSTE GANG

In de voorafgaande paragraaf bevatte het uitvoerbestand exact dezelfde gegevens als het invoerbestand. Alleen de volgorde verschilde. Veronderstel dat we voor de sorteerbewerking records willen wijzigen, verwijderen of toevoegen.

Om dit te volbrengen moeten we onze eigen codering invoeren in het logische patroon van de eerste gang van de sorteerbewerking. We hoeven dan niet een apart programma te maken dat het gesorteerde bestand in een gewijzigd bestand omzet. Dit kan worden bereikt door de specificatie INPUT PROCEDURE IS section-name in plaats van de USING-passage in de sort-zin. Section-name is de naam van een section in de Procedure Division die onze eigen codering voor de eerste gang bevat.

In deze section lezen we het invoerbestand op de gebruikelijke niet-sort wijze in. We maken het gewijzigde of verbeterde record klaar in het gegevensrecord van het sorteringsbestand. Wanneer het record klaar is, coderen we RELEASE record-name, waarbij record-name de naam is van het record van het sorteringsbestand. Indien we een record willen verwijderen, volgt er eenvoudig geen release-opdracht.

Laten we ons voorstellen dat het in het voorbeeld van paragraaf R3 gewenst is alleen voor werknemers die meer dan 40 uur gewerkt hebben een gesorteerd uitvoerbestand klaar te maken. De records voor alle andere werknemers moeten worden weggelaten. Met dezelfde beschrijvingen voor de Data Division kunnen we dit bereiken door door te schrijven:

```
PROCEDURE DIVISION.
A1-HOOFD SECTION.
A1.
    OPEN INPUT TIJD-KAARTEN.
    SORT WERKBESTAND ON
         ASCENDING KEY UREN WERKNEMERS-NR
         INPUT PROCEDURE IS B1-EERSTE-GANG-CODERING
         GIVING GESORTEERDE-TIJD-KAARTEN.
    CLOSE TIJD-KAARTEN.
    STOP RUN.
B1-EERSTE-GANG-CODERING SECTION.
B1.
    PERFORM C1-VERWERK-KAART-IN UNTIL EVB-TIJD-KAARTEN.
```

```
C1-VERWERK-KAART-IN SECTION.
C1.
    READ TIJD-KAARTEN AT END
       Set EVB-TIJD-KAARTEN true.
    IF NOT EVB-TIJD-KAARTEN
       MOVE KAART-IN TO GEWERKTE-TIJD
       IF UREN GREATER THAN 40
          RELEASE GEWERKTE-TIJD.
```

Een betere methode bestaat in het beschrijven van het veld UREN in het invoerrecord en het onderzoeken van dit invoerrecord voordat het verplaatst is naar het record van het sorteringsbestand. Er is namelijk een voorziening waardoor we kunnen coderen RELEASE GEWERKTE-TIJD FROM KAART-IN; de MOVE-instructie is dan overbodig geworden.

Opgave

1. Wijzig uw antwoord op vraag 1, paragraaf R3 zo, dat records met alleen maar blanco posities niet in het uitvoerbestand worden opgenomen.

R5 EIGEN CODERING – LAATSTE GANG

Evenals we nog bewerkingen kunnen uitvoeren met records tijdens de eerste gang doordat we het vrijgeven van de records met RELEASE kunnen besturen, zo kunnen we dat ook tijdens de laatste gang door middel van de opdracht RETURN. In dit geval specificeren we OUTPUT PROCEDURE IS section-name in plaats van de GIVING-passage in de sort-zin.

Veronderstel dat het in het voorbeeld van paragraaf R3 wenselijk is niet alleen een compleet gesorteerd bestand met tijdkaartenrecords op te zetten, maar ook die records uit te lijsten waarin het aantal gewerkte uren de 40 overschrijdt. Aangenomen dat we een passend afdrukbestand en record hebben beschreven, kunnen we coderen:

```
    OPEN OUTPUT GESORTEERDE-TIJD-KAARTEN AFDRUKBESTAND.
    SORT WERKBESTAND ON
           ASCENDING KEY UREN WERKNEMERS-NR
           USING TIJD-KAARTEN
           OUTPUT PROCEDURE IS B1-LAATSTE-GANG-CODERING.
    CLOSE  GESORTEERDE-TIJD-KAARTEN AFDRUKBESTAND.
    STOP RUN.
B1-LAATSTE-GANG-CODERING SECTION.
```

```
B1.
    PERFORM C1-VERWERK-GEWERKTE-TIJD UNTIL EVB-WERKBESTAND.
C1-VERWERK-GEWERKTE-TIJD SECTION.
C1.
    RETURN WERKBESTAND
    AT END
       Set EVB-WERKBESTAND true.
    IF NOT EVB-WERKBESTAND
       WRITE RECORD-UIT FROM GEWERKTE-TIJD
       IF UREN GREATER THAN 40
          WRITE REGEL FROM GEWERKTE-TIJD.
```

Opgave

1. Bij RELEASE moet de recordnaam van het sorteringsbestand worden vermeld, en bij RETURN de bestandsnaam van het sorteringsbestand. Waarom?

R6 DE OPDRACHT MERGE

De opdracht MERGE stelt ons in staat twee of meer bestanden die al al in dezelfde volgorde staan, samen te voegen tot één nieuw bestand.

De opdracht MERGE werkt met een sorteringsbestand op dezelfde wijze als SORT. De syntaxis van beide opdrachten in nagenoeg identiek, behalve dat MERGE geen eigen codering bij de eerste gang kan hebben. Het complete formaat staat in Appendix D.

ANTWOORDEN HOOFDSTUK R

Paragraaf R1

1. Het record met werknemersnummer 1100.

Paragraaf R3

1.
```
FILE SECTION.
SD  SORTEERBESTAND.
01  BOEKREC.
    02 AUTEURSNAAM    PICTURE X(20).
    02 CLASSIFICATIE  PICTURE X(10).
    02 BOEKTITEL      PICTURE X(70).
```

```
FD  BOEKEN-IN.
01  BOEK-IN PICTURE X(100).
FD  BOEKEN-UIT.
01  BOEK-UIT PICTURE X(100).
PROCEDURE DIVISION.
A1.
    SORT SORTEERBESTAND ON
         ASCENDING KEY CLASSIFICATIE AUTEURSNAAM BOEKTITEL
         USING BOEKEN-IN
         GIVING BOEKEN-UIT.
    STOP RUN.
```

2. Waarschijnlijk niet. Het bestand zal staan in de door de computer opgelegde sorteervolgorde. De bibliothecaris zal waarschijnlijk willen dat het bestand wordt gerangschikt overeenkomstig de indexeringsregels van zijn bibliotheek. Deze laatste doen allerlei dingen die de computer niet doet, bijvoorbeeld het 'verwaarlozen' van spaties, het uitbreiden van afkortingen tot volledige woorden (Dr. wordt onder 'doctor' opgenomen), het vertalen van getallen in woorden ('2001 - een odyssee door de ruimte' wordt onder 'tweeduizendeneen' opgenomen).

Paragraaf R4

1.

```
        SORT SORTEERBESTAND ASCENDING KEY etc.
             INPUT PROCEDURE IS B1-LAAT-SPATIES-WEG
             GIVING BOEKEN-UIT.
        .
        .
        .
    B1-LAAT-SPATIES-WEG SECTION.
    B1.
        Until evb-boeken-in
              Read boeken-in record
              at end
                 Set evb-boeken-in true
              else
                 if classificatie not = spaces
                    RELEASE BOEKREC FROM BOEK-IN.
```

Paragraaf R5

1. Zie paragraaf G2, opgave 3.

S DE REPORT WRITER

S1 ACHTERGROND

De report writer gebruikt principes die nogal verschillen van de meeste andere COBOL-principes. COBOL-leergangen noemen het om deze reden dikwijls niet, waardoor er maar weinig programmeurs zijn die het gebruik van de report writer kennen.

Het belangrijkste verschil is dat de report writer in hoofdzaak een *declaratieve* taal is, dat wil zeggen de programmeur declareert welke resultaten hij wenst en laat het vervolgens aan de machine over hoe deze resultaten te bereiken. Andere aspecten van COBOL zijn hoofdzakelijk *procedureel*, dat wil zeggen dat de programmeur exact moet aangeven *hoe* de resultaten worden bereikt. Ofschoon het duidelijk lijkt dat de declaratieve stijl wenselijk is, blijkt deze in de praktijk niet erg soepel te zijn. De report writer is heel geschikt voor rechttoe-rechtaan rapporten, maar neigt ertoe tamelijk ingewikkeld te worden voor iets wat buiten het gewone stramien valt.

De report writer heeft zoveel faciliteiten dat de volledige behandeling daarvan in detail het leren van een nieuwe programmeertaal zou inhouden. Het is mijn opzet een basis te verschaffen waarop de student desgewenst kan steunen bij het bestuderen van de handleidingen van de fabrikant.

De report writer beoogt het produceren van overzichten op de regeldrukker te vergemakkelijken. De waarde van de report writer berust op de identieke vorm van veel overzichten. Ze beginnen met *kopregels*, gevolgd door *detailregels*, en *totaalregels* bij iedere *controle-overgang*. Het schrijven van het overzicht in conventioneel COBOL vereist het programmeren van standaardprocedures voor het nummeren van de pagina's, het tellen van de regels, het herhalen van de kopregels op iedere nieuwe pagina, het voorzien in spaties tussen de verschillende gegevenselementen. De report writer behandelt al deze dingen automatisch met slechts een minimum uitdrukkelijke opdrachten van de programmeur.

S2 VOORBEELD VAN EEN OVERZICHT, FD VAN EEN OVERZICHTSBESTAND

Veronderstel dat onze taak is het samenstellen van een verkoopoverzicht op basis van verkooprecords die de volgende informatie bevatten:

```
Kol   1     Gebiedscode
      2-5   Vertegenwoordigersnummer
      6-11  Verkoopdatum DDMMJJ
     12-17  Verkoopbedrag 9(4)V99.
```

Het bestand met records voor een maand is in opklimmende volgorde gerangschikt naar gebied en binnen een gebied naar vertegenwoordigersnummer. Het gewenste overzicht ziet er als volgt uit:

```
         MAANDELIJKS VERKOOPOVERZICHT          PAGINA  1

GEBIED   VERTEGENWOORDIGER        VERKOOPDATUM  BEDRAG
  1           1001                01 01 83       27,46
  1           1001                07 01 83      100,00
         TOTAAL VERTEGENWOORDIGER               127,46
  1           1002                01 01 83       50,--
  1           1002                21 01 83      100,00
         TOTAAL VERTEGENWOORDIGER               150,00
         TOTAAL GEBIED                          277,46
  2      etc..etc..
         .
         .
         .
         TOTAAL-GENERAAL                       9900,46
```

Er zijn drie controleovergangen in dit overzicht – verandering van vertegenwoordigersnummer, verandering in gebiedscode en einde van het overzicht.

Indien het overzicht moet worden geproduceerd door de report writer, zal de File-section van het programma als volgt luiden:

```
FILE SECTION.
FD  AFDRUKBESTAND REPORT IS VERKOOPOVERZICHT.
FD  VERKOOPBESTAND VALUE OF IDENTIFICATION "VERKOOP".
01  VERKOOPRECORD.
    02 GEBIEDSCODE            PICTURE 9.
    02 VERTEGENWOORDIGERSNR   PICTURE 9(4).
    02 VERKOOPDATUM           PICTURE 9(6).
    02 VERKOOPWAARDE          PICTURE 9(4)V99.
```

Merk op dat AFDRUKBESTAND een passage: REPORT IS heeft. Het overzicht met de door de programmeur bedachte naam VERKOOP-OVERZICHT wordt beschreven in de Report Section.

S3 DE REPORT SECTION

De vormgeving van het overzicht en de controleovergangen worden beschreven in de Report Section, de laatste section in de DATA DIVISION. De omschrijving omvat een RD-passage (report description), die de controleovergangen specificeert, gevolgd door beschrijvingen op 01-niveau die de regels van het overzicht specificeren.

De report description in ons voorbeeld kan zijn:

```
REPORT SECTION.
RD  VERKOOPOVERZICHT
    CONTROLS ARE FINAL  GEBIEDSCODE  VERTEGENWOORDIGERSNR
    PAGE LIMIT 55 LINES.
```

De passage CONTROLS ARE beschrijft de controleovergangen in de volgorde van buiten naar binnen, dat wil zeggen de belangrijkste controleovergang komt het eerst (degene die het laatst iets afdrukt), de minst belangrijke het laatst (de overgang die het eerst iets afdrukt). FINAL is het sleutelwoord voor de controleovergang die optreedt aan het eind van het overzicht. PAGE LIMIT vertelt de report writer op een nieuwe pagina over te gaan en kopregels af te drukken wanneer het gespecificeerde aantal regels is bereikt.

De RD-beschrijving wordt gevolgd door 01-beschrijvingen voor de regels. Deze beschrijvingen verschillen van de gebruikelijke 01-beschrijvingen in de hierna volgende opzichten:

1. Ze kunnen een TYPE krijgen. TYPE PAGE-HEADING betekent dat de regel aan de kop van de pagina staat en automatisch op iedere nieuwe pagine wordt afgedrukt. TYPE DETAIL houdt in dat het om een detailregel gaat die wordt afgedrukt door opdrachten in de Procedure Division. TYPE CONTROL-FOOTING gevolgd door de gegevensnaam van een controleovergang betekent een totaalregel, die automatisch wordt afgedrukt zodra deze controleovergang plaatsvindt.
2. Namen van records en velden kunnen worden weggelaten. Alleen de detailregel heeft in het algemeen een naam nodig.
3. De passage LINE specificeert hoe het papier moet worden opgeschoven. LINE 1 betekent: druk af op de eerste regel, LINE PLUS 1 betekent: druk af één regel na de voorafgaande regel, LINE PLUS 2 betekent: druk af twee regels na de voorafgaande regel, enz.
4. De passage COLUMN specificeert in welke kolom van de pagina het veld dient te beginnen. Fillers zijn niet nodig; de report writer voegt automatisch spaties in tussen de velden.
5. Het is niet noodzakelijk gegevens naar de velden te verplaatsen. De passage SOURCE wordt gebruikt om vast te leggen waarvandaan de gegevens komen.
6. Totalen kunnen worden geaccumuleerd in een veld door de passage SUM.

De beschrijving van ons voorbeeld laat het gebruik van deze kenmerken in de praktijk zien.

```
01  TYPE PAGE HEADING.
    02 LINE 1.
       03 COLUMN 9  PICTURE X(28)
          VALUE "MAANDELIJKS VERKOOPOVERZICHT".
       03 COLUMN 46 PICTURE X(6) VALUE "PAGINA".
       03 COLUMN 53 PICTURE Z9 SOURCE PAGE-COUNTER.
    02 LINE PLUS 2.
       03 COLUMN 1  PICTURE X(40) VALUE "GEBIED
       " VERTEGENWOORDIGER              VERKOOPDATUM  BEDRAG".
01  VERKOOP-REGEL TYPE DETAIL LINE PLUS 1.
    02 COLUMN 3  PICTURE 9 SOURCE GEBIEDSCODE.
    02 COLUMN 14 PICTURE 9(4) SOURCE VERTEGENWOORDIGERSNR.
    02 COLUMN 35 PICTURE 99B99B99 SOURCE VERKOOPDATUM.
    02 COLUMN 48 PICTURE ZZZ9.99 SOURCE VERKOOPWAARDE.
01  TYPE CONTROL FOOTING VERTEGENWOORDIGERSNR
    LINE PLUS 1.
    02 COLUMN 9  PICTURE X(24)
                 VALUE "TOTAAL VERTEGENWOORDIGER".
    02 COLUMN 47 PICTURE Z(4)9.99 SUM VERKOOPWAARDE.
01  TYPE CONTROL FOOTING GEBIEDSCODE
    LINE PLUS 1.
    02 COLUMN 9  PICTURE X(13) VALUE "TOTAAL GEBIED".
    02 COLUMN 46 PICTURE Z(5)9.99 SUM VERKOOPWAARDE.
01  TYPE CONTROL FOOTING FINAL
    LINE PLUS 1.
    02 COLUMN 9  PICTURE X(15) VALUE "TOTAAL-GENERAAL".
    02 COLUMN 45 PICTURE Z(6)9.99 SUM VERKOOPWAARDE.
```

1. PAGE COUNTER is een gereserveerd woord voor de paginateller, die automatisch door de report writer wordt bijgehouden.
2. Het eerste record beschrijft twee regels die moeten worden afgedrukt op de kop van de pagina.
3. Het tweede record beschrijft de detailregel, die wordt afgedrukt onder besturing van de opdrachten in de Procedure Division (zie volgende paragraaf).
4. De laatste drie records beschrijven de regels die worden afgedrukt bij de gespecificeerde controleovergangen, dat wil zeggen verandering van vertegenwoordiger, verandering van gebied en einde van het overzicht.

De passage SUM houdt in dat het totaal van alle voorafgaande VERKOOPWAARDEn die zijn afgedrukt moet worden bijgehouden in het desbetreffende veld, dat wil zeggen: steeds wanneer een detailregel is afgedrukt, wordt VERKOOPWAARDE toegevoegd aan de velden die beginnen in de kolommen 47, 46 en 45 respectievelijk in de laatste drie records. Dit doorkruist de algemene regel, dat geen rekenkundige bewerkingen mogen worden uitgevoerd met een opge-

maakt element, maar dit wordt door de report writer in orde gemaakt. Deze velden worden automatisch op nul gezet wanneer de regels waarin ze staan zijn afgedrukt.

Opgave

1. In de passage SUM kan ook worden verwezen naar een totaalveld waarin voor de voorafgaande controleovergang een totaal is bijgehouden. Dat veld moet dan natuurlijk wel een naam hebben gekregen. Dit kan een hoop optellingen besparen voor het bijhouden van de totalen van de hogere niveaus. Wijzig het voorbeeld in deze paragraaf zodanig dat het 'totaal gebied' ontstaat door sommatie van de 'totaal vertegenwoordiger's en het 'totaal-generaal' door sommatie van de 'totaal gebied'en. Tref ook voorzieningen waardoor de totaalregels afgedrukt worden met respectievelijk dubbele, drievoudige en viervoudige interlinie

S4 OPDRACHTEN IN DE PROCEDURE DIVISION

Er bestaan drie specifieke opdrachten om het overzicht te verwerken. Deze zijn:

1. INITIATE report-name. Deze opdracht zorgt ervoor dat de paginateller zijn beginwaarde krijgt en dat de kop op de eerste pagina wordt afgedrukt.
2. GENERATE record-name: regelt het afdrukken van een detailregel (alle andere regels worden automatisch afgedrukt).
3. TERMINATE report-name: regelt de afhandeling van de laatste controleovergang. Dit telt als een controleovergang voor alle controlevelden.

Toepassing van deze opdrachten op ons voorbeeld levert de volgende Procedure Division op:

```
PROCEDURE DIVISION.
A1-PRODUCEER-VERKOOPOVERZICHT.
    OPEN OUTPUT AFDRUKBESTAND.
    INITIATE VERKOOPOVERZICHT.
    OPEN INPUT VERKOOPBESTAND.
    PERFORM B1-VERWERK-VERKOOPRECORD UNTIL EVB-VERKOOPBESTAND.
    CLOSE VERKOOPBESTAND.
    TERMINATE VERKOOPOVERZICHT.
    CLOSE AFDRUKBESTAND.
    STOP RUN.
B1-VERWERK-VERKOOPRECORD.
    READ VERKOOPBESTAND
        AT END
            set EVB-VERKOOPBESTAND true
        else
            GENERATE VERKOOP-REGEL.
```

Dit zijn alle opdrachten benodigd om het overzicht zoals is beschreven in paragraaf S2 af te drukken.

Er is geen opgave bij deze paragraaf, maar het zou leerzaam zijn een programma te schrijven voor het afdrukken van het overzicht, maar dan zonder gebruikmaking van de report writer.

ANTWOORDEN HOOFDSTUK S

Paragraaf S3

1.
```
01  TYPE CONTROL FOOTING  VERTEGENWOORDIGERSNR
    LINE PLUS 2.
    02 COLUMN 9  PICTURE X(24)
                 VALUE "TOTAAL VERTEGENWOORDIGER".
    02 TOTVERT COLUMN 47 PICTURE Z(4)9.99 SUM VERKOOPWAARDE.
01  TYPE CONTROL FOOTING GEBIEDSCODE
    LINE PLUS 3.
    02 COLUMN 9  PICTURE X(13) VALUE "TOTAAL GEBIED".
    02 TOTGEB COLUMN 46 PICTURE Z(5)9.99 SUM TOTVERT.
01  TYPE CONTROL FOOTING FINAL
    LINE PLUS 4.
    02 COLUMN 9  PICTURE X(15) VALUE "TOTAAL-GENERAAL".
    02 COLUMN 45 PICTURE Z(6)9.99 SUM TOTGEB.
```

Wanneer u wilt dat dubbele respectievelijk drievoudige interlinie zowel voor als na de eerste respectievelijk de tweede totaalregel komt, kunt u dit bereiken door te coderen na 02 TOTVERT..

```
02 LINE PLUS 1.
```

en na 02 TOTGEB...

```
02 LINE PLUS 2.
```

De automatische overgang op nieuwe regel voor de *volgende* af te drukken regel zorgt dan voor een dubbele respectievelijk drievoudige interlinie.

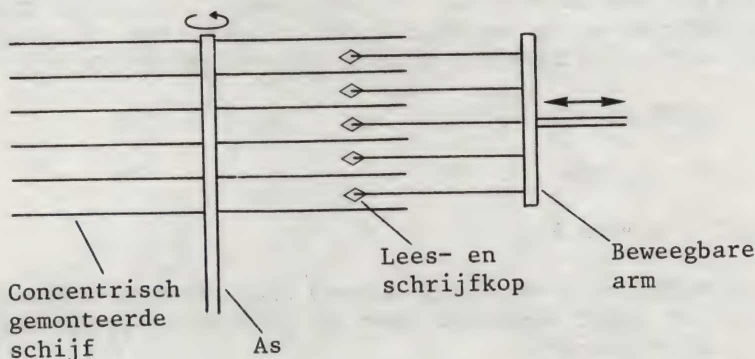
Als slechts twee regels wit worden gewenst tussen de regel 'totaal vertegenwoordiger' en de regel 'totaal gebied', moet het vooruitgaan van de regel 'totaal gebied' worden verminderd van 3 naar 2. Evenzo moet het vooruitgaan van de regel 'totaal-generaal' worden verminderd van 4 naar 2.

T DIRECT-TOEGANKELIJKE BESTANDEN

T1 DE MAGNEETSCHIJFEENHEID EN HET SCHIJVENPAKKET

Een schijfeenheid legt informatie vast op het oppervlak van een schijf. De aard van de in de praktijk aan te treffen schijven verschilt aanzienlijk en loopt van de buigzame schijf van 5¼ inch, die enige tienduizenden tekens kan bevatten, tot de harde veelvoudige schijven, bevestigd aan een gemeenschappelijke as met een capaciteit van in totaal honderden miljoenen tekens. In dit boek staan ongeveer een half miljoen tekens.

Figuur 47 illustreert een veelvoudig schijvenssysteem met lees- en schrijfkoppen die zijn bevestigd aan een beweegbare zoekarm. Sommige systemen hebben een vaste arm en leveren een lees/schrijfkop voor ieder spoor op de schijf.



FIGUUR 47 Het schijvenpakket.

Terwijl de schijf draait, worden de gegevens door middel van de schrijfkop overgebracht en magnetisch vastgelegd op het oppervlak in een cirkelvormig spoor. Schijfeenheden bezitten in het algemeen tussen de 80 en een paar honderd sporen. Bij een veelvoudig schijvensysteem worden de sporen die toegankelijk zijn op alle schijven zonder de zoekarm te bewegen, een cilinder genoemd.

Bij een schijfeenheid is ieder record direct toegankelijk, mits zijn positie bekend is. Dit gaat als volgt:

1. Beweeg de kam van lees/schrijfkoppen naar de juiste positie (dit wordt een zoekactie (Engels: seek) genoemd. Deze fase wordt weggelaten in het geval van een schijfeenheid met een kop per spoor.
2. Selecteer het vereiste spoor (een kop-switch genaamd). Deze fase is niet nodig in het geval van een enkelvoudige schijfeenheid.
3. Wacht totdat het gewenste record van de draaiende schijf zich onder de lees/schrijfkop bevindt (zoekbewerking; Engels: search).
4. Breng het record met gegevens over.

Deze fysieke bewerkingen worden automatisch door het besturingssysteem uitgevoerd. Zoals straks zal worden toegelicht, zijn deze bewerkingen bezien vanuit de programmalogica niet van belang voor de programmeur. Maar van programmeurs wordt wel vaak verwacht dat ze kunnen schatten hoeveel tijd er gaat zitten in het ophalen van records op bestanden en daarvoor is kennis van de kenmerken van de opslagapparatuur een vereiste.

Opgaven

1. Hoeveel cilinders bevinden zich in een schijvenpakket van 200 sporen met bewegende koppen?
2. Hoeveel gegevens kunnen ongeveer worden opgeslagen op een cilinder van een schijvenpakket met 6 schijven (10 oppervlakken) en met 8000 tekens per spoor?
3. Hoeveel gegevens kunnen ongeveer worden opgeslagen op een schijvenpakket van 6 schijven met 400 cilinders en 8000 tekens per spoor?
4. Worden de gegevens op de binnenste sporen met een grotere, kleinere of dezelfde dichtheid vastgelegd als de gegevens op de buitenste sporen?

T2 SEQUENTIËLE BESTANDSORGANISATIE

Soms wordt onderscheid gemaakt tussen een sequentieel georganiseerd bestand, waarin de records worden opgeslagen in de volgorde gedicteerd door in een sleutelveld van het record vastgelegde gegevens, en een serieel georganiseerd bestand, waarin de records in geen enkele sleutelvolgorde staan. In COBOL bestaat er tussen deze twee geen verschil. Hoewel het verschil de logica van het pro-

gramma kan beïnvloeden, zijn beide gevallen sequentiële bestanden in COBOL, hetgeen inhoudt dat de enige volgorde waarin records kunnen worden teruggelezen dezelfde volgorde is als de volgorde waarin ze werden geschreven.

Bij een sequentieel bestand worden de records achter elkaar op een spoor vastgelegd. Wanneer het einde van een spoor wordt bereikt, wordt het vastleggen in het algemeen vervolgd op het direct eronder gelegen spoor, dat wil zeggen in dezelfde cilinder. Deze methode vermindert de zoekactiviteit, die tijdrovend kan zijn. Een zoekactie naar de volgende cilinder vindt alleen plaats wanneer de eerste cilinder volledig met gegevens is gevuld, enz. Een zoekactie naar een aangrenzende cilinder neemt gewoonlijk enkele milliseconden in beslag. Verschillende systemen kunnen de gegevens met meer dan een schrijfkop tegelijk vastleggen, waardoor de gegevensoverdracht versneld wordt.

Er kunnen verscheidene bestanden op één schijvenpakket worden vastgelegd (mits de bestanden natuurlijk klein genoeg zijn). Gewoonlijk laat men ze ieder een bepaald aantal cilinders bezetten, dat wil zeggen dat het bestand begint bij het eerste record op het eerste spoor van de eerste cilinder die aan dat bestand is toegewezen, terwijl het fysieke einde van het bestand staat op het laatste record van het laatste spoor van de laatste toegewezen cilinder. Indien de eigenlijke bestandsgegevens minder cilinders vereisen dan daaraan zijn toegewezen, zal het logische einde van het bestand dichter bij het begin staan dan het fysieke einde van het bestand. (De terminologie van bestandstoewijzing verschilt aanzienlijk, waarbij de eenheid van toewijzing afwisselend een area (gebied), page (pagina) of extent (uitbreiding) wordt genoemd. Soms stemt deze eenheid niet overeen met de omvang van een spoor of cilinder. Soms kan de omvang van de eenheid van toewijzing door de programmeur worden geregeld.)

Als records worden toegevoegd aan het einde van het bestand (dit kan in COBOL door te declareren: OPEN EXTEND bestandsnaam alvorens weg te schrijven naar het bestand), kan het bestand meer ruimte nodig hebben dan dat werd toegewezen. Bij de meeste computersystemen zal het besturingssysteem automatisch meer ruimte toewijzen. De methode om het besturingssysteem te vertellen hoeveel ruimte in eerste aanleg te reserveren en hoeveel extra ruimte te reserveren steeds wanneer de vorige toewijzing vol is, is niet standaard. De meeste systemen stellen de programmeur in staat de maximumruimte die bezet mag worden door het bestand voor te schrijven. Als deze ruimte wordt overschreden, ontstaat een foutconditie.

De File-control beschrijving voor een sequentieel bestand op schijf luidt:

```
FILE-CONTROL.
    SELECT VOORBEELD-SCHIJF-BESTAND ASSIGN TO DISK.
```

Het sleutelwoord DISK kan verschillen per fabrikant.

Een schijfbestand op deze wijze beschreven wordt precies zo verwerkt alsof het een magneetband is, behalve bij het 'ter-plaatse-wijzigen', zoals we nu zullen bespreken.

Een kenmerk van schijfbestanden is dat het bestand kan worden bijgewerkt door het overschrijven van de bestaande records - dit noemen we *ter plaatse wijzigen*. Dit kan voordelig zijn in termen van verwerkingstijd en programmeringsgemak, maar het is nadelig in termen van zekerheid, aangezien de vorige versie van het record wordt vernietigd. (Dit nadeel kunnen we ondervangen door verschillende methoden waarbij de eerdere en latere versies van de records op een ander apparaat worden gekopieerd.)

Voor het volvoeren van een wijziging ter plaatse moeten we na een read-opdracht en na wijziging van het record de opdracht REWRITE gebruiken. Wanneer een read-opdracht gegeven wordt, herinnert het COBOL-systeem zich namelijk de plaats waar het gelezen record stond; wanneer daarna (bij een sequentieel bestand) een rewrite-opdracht volgt, wordt het record weggeschreven naar dezelfde plaats. Als de records van een sequentieel bestand ter plaatse worden gewijzigd en er dus zowel read- als rewrite-opdrachten zullen plaatsvinden, wordt dit bestand I-O geopend (zowel voor input als voor output).

Een voorbeeld kan wellicht helpen om deze theorie inhoud te geven. Veronderstel dat er een sequentieel bestand bestaat, dat records van 80 tekens bevat. Als een record alleen maar nullen bevat, moet het worden vervangen door een record met de tekens **ALLEMAAL NULLEN**.

```
:
FILE-CONTROL.
    SELECT SCHIJFBESTAND ASSIGN TO DISK.
:
FD  SCHIJFBESTAND.
01  SCHIJFREC PICTURE X(80).
:
PROCEDURE DIVISION.
A1-VERWERK-SCHIJFBESTAND.
    OPEN I-O SCHIJFBESTAND.
    perform until einde-schijfbestand
    READ SCHIJFBESTAND
    AT END
        set einde-schijfbestand true
    not end
       MOVE "**ALLEMAAL NULLEN**" TO SCHIJFREC
       REWRITE SCHIJFREC.
```

We vatten de verwerking van sequentiële bestanden op schijf samen:

OPEN OUTPUT - een WRITE-opdracht schrijft het record naar de volgende sequentiële plaats

OPEN INPUT - een READ-opdracht haalt het record van de volgende sequentiële plaats op

OPEN I-O - een READ-opdracht haalt het record van de volgende sequentiële plaats op; een REWRITE-opdracht schrijft het record terug naar de plaats van het laatst gelezen record.

OPEN EXTEND - een WRITE-opdracht schrijft het record naar de volgende sequentiële plaats achter het voorgaande einde van het bestand.

T3 DIRECTE TOEGANG DOOR RELATIVE KEY

Bij deze methode kunnen we toegang verkrijgen tot een record door het rangnummer van het record in het bestand te vermelden, waarbij we tellen vanaf (relatief ten opzichte van) het begin van het bestand. Dit 'relatieve recordnummer' (rangnummer) wordt geplaatst in een RELATIVE KEY gegevenselement in de working storage. Aangezien het operating system op de hoogte is van de bestandsgrenzen en de omvang van de records, berekent het uitgaande van het relatieve recordnummer automatisch het fysieke adres voor het ophalen of opslaan van het record.

Bij het gebruik van relative keys is het relatieve recordnummer (of een ander gegeven waar dit relatieve recordnummer van afgeleid kan worden) gewoonlijk als een gegevenselement in de records van het bestand opgenomen, maar dit is niet noodzakelijk. De File-control-paragraaf kan er als volgt uitzien:

```
FILE-CONTROL.
    SELECT WERKNEMERSBESTAND ASSIGN TO DISK
    ORGANIZATION IS RELATIVE
    ACCESS MODE IS RANDOM
    RELATIVE KEY IS RECORD-NUMMER.
```

Merk op dat ORGANIZATION moet worden gespeld met een Z. Als ACCESS MODE IS RANDOM niet wordt gedeclareerd, neemt de compiler aan, dat het bestand sequentieel in volgorde van het relatieve recordnummer moet worden gelezen. De ORGANIZATION van het bestand is een vaste, fysieke eigenschap die niet kan worden veranderd, nadat het bestand is gemaakt, door welke declaratie in het programma dan ook. De ACCESS MODE kan per programma variëren, mits deze verenigbaar is met de ORGANIZATION van het bestand (u kunt bijvoorbeeld niet een sequentieel georganiseerd bestand voor willekeurige toegankelijkheid (access mode is random) declareren). Men zou kunnen denken dat ACCESS MODE, omdat het

iets lokaals in het programma is, beter kan worden gedeclareerd in de File Definition dan in File-Control, alwaar de ORGANIZATION-passage de eigenschap specificeert die globaal is voor alle programma's; deze mening wordt nu ook door CODASYL gedeeld, maar ze is nog niet gestandaardiseerd.

Een ontwerp van de Data Division voor dit bestand kan er zo uitzien:

```
FILE SECTION.
FD  WERKNEMERSBESTAND.
01  WERKN-RECORD.
    02 SALARISADMINISTRATIE-NUMMER PICTURE 9(5).
    :
    .
WORKING-STORAGE SECTION.
:
.
    02 RECORD-NUMMER PICTURE 9(5) COMP SYNC RIGHT.
```

Een bestand dat RELATIVE is georganiseerd en met ACCESS MODE IS RANDOM, kan worden geopend voor invoer (READ-opdrachten geoorloofd); uitvoer (WRITE-opdrachten geoorloofd); invoer-uitvoer (READ-, REWRITE- en DELETE-opdrachten geoorloofd – waarbij de laatste opdracht logischerwijze het record uit het bestand weghaalt zodat het niet door een latere READ-opdracht kan worden opgehaald). De passage INVALID KEY is beschikbaar bij al deze opdrachten om verschillende foutcondities die kunnen ontstaan op te sporen (bijvoorbeeld een poging om een READ- of DELETE-opdracht uit te voeren van records die er niet zijn).

Stel dat er 1000 records in WERKNEMERSBESTAND zitten, overeenkomend met de SALARISADMINISTRATIE-NRs 10.000 t/m 11.000. Een programma moet van de operator een salarisadministratienummer ontvangen (en dit opslaan in W-SAL-NR), het overeenkomstige werknemersrecord opzoeken en dit afbeelden. Een ontwerp van de opdrachten in de Procedure Division kan er als volgt uitzien:

```
OPEN INPUT WERKNEMERSBESTAND.
:
.
ACCEPT W-SAL-NR.
SUBTRACT 10000 FROM W-SAL-NR GIVING RECORD-NUMMER.
READ WERKNEMERSBESTAND
INVALID KEY
    afbeelden "ONGELDIG SALARISADMINISTRATIENUMMER"
    andere fouthandelingen uitvoeren
not invalid
    uitvoer van WERKN-RECORD klaarmaken
    uitvoer afbeelden.
```

Zoals reeds vermeld is het mogelijk relatieve bestanden sequentieel (in volgorde van relatief recordnummer) te verwerken. Het is ook mogelijk zo'n bestand dynamisch te verwerken (mengsel van

sequentiële en directe toegangstechniek). Deze voorzieningen zijn dezelfde als die worden beschreven voor geïndiceerde bestanden in de volgende paragraaf.

T4 GEÏNDICEERDE BESTANDEN

Waarschijnlijk is het geïndiceerde bestand (dikwijls index-sequentieel genoemd) de meest verbreide vorm van directe bestandsorganisatie. De reden hiervan is, dat het bestand zowel sequentieel (in volgorde van een gegevenselement dat een sleutelveld is van het record, welke volgorde verschillend mag zijn van de volgorde waarin de records werden opgeslagen) als direct (door het specificeren van de sleutelwaarde van het gewenste record) toegankelijk kan worden gemaakt.

Kort gezegd wordt het fysieke adres van een record waarvan de sleutel bekend is, door de computer opgespoord door middel van een hiërarchie van indices. De technische realisatie hiervan verschilt lichtelijk per fabrikant en behoeft ons nu niet bezig te houden; belangrijk voor de programmeur is dat hij het gehele bestand kan verwerken in volgorde van record key alsof het een gesorteerd bestand op magneetband was. Het probleem hoe de gegevens zijn opgeslagen wordt automatisch afgehandeld.

Om een geïndiceerd bestand sequentieel te verwerken in volgorde van record key kunnen de volgende File-Control-opdrachten verschijnen:

```
SELECT WERKNEMERSBESTAND ASSIGN TO DISK
ORGANIZATION IS INDEXED
ACCESS MODE IS SEQUENTIAL
RECORD KEY IS WERKNEMERS-NUMMER.
```

WERKNEMERS-NUMMER is de naam van een veld in de recordbeschrijving van het WERKNEMERSBESTAND. Als ACCESS MODE IS SEQUENTIAL is gespecificeerd, kan het geïndiceerde bestand worden gemaakt door het schrijven van records in volgorde van record key (het bestand moet geopend zijn voor OUTPUT) of ter plaatse worden bijgewerkt door REWRITE- of DELETE-opdrachten (geopend voor I-O). READ krijgt nu de passage AT END; de andere opdrachten krijgen de passage INVALID KEY.

Het is ook mogelijk de sequentiële verwerking te beginnen bij een andere positie dan het eerste record in het bestand door middel van de opdracht START. Indien bijvoorbeeld in het geval hierboven WERKNEMERS-NUMMER een groepsveld is dat een afdelingsnummer van 2 cijfers en een volgnummer van 4 cijfers bevat en we willen de verwerking beginnen bij afdelingsnummer 10, dan kunnen we coderen:

```
FD  WERKNEMERSBESTAND.
.
.
.
    02 WERKNEMERS-NUMMER.
       03 AFDELING PICTURE 99.
       03 WERKN-VOLGNR PICTURE 9999.
    .
    .
    .
    OPEN INPUT WERKNEMERSBESTAND.
    MOVE "100000" TO WERKNEMERS-NUMMER.
    START WERKNEMERSBESTAND
    KEY GREATER THAN WERKNEMERS-NUMMER
    INVALID KEY
       DISPLAY "GEEN AFDELINGSNR GROTER DAN 09"
       verdere afhandeling van fout
    not invalid
       repeat until einde-werknemersbestand
          READ WERKNEMERSBESTAND RECORD
          AT END
             set einde-werknemersbestand true
          not end
             verwerk record.
```

De eerste uitvoering van de read-opdracht zal het eerste record in het bestand dat voldoet aan de gespecificeerde KEY-voorwaarde ophalen. Merk op dat in de KEY-passage de gegevensnaam moet staan die gespecificeerd is als de RECORD KEY in de FILE-CONTROL paragraaf (waarom dit zo is zal duidelijk worden in paragraaf T5). De passage INVALID KEY in de START-opdracht wordt uitgevoerd indien geen enkel record in het bestand voldoet aan de gespecificeerde voorwaarde.

Om directe toegang tot een geïndiceerd bestand te verkrijgen, dat wil zeggen dat we een record uit bijvoorbeeld het WERKNEMERS-BESTAND direct inlezen door het specificeren van de gewenste record key in WERKNEMERS-NUMMER, zal de paragraaf File-Control omvatten:

```
SELECT WERKNEMERSBESTAND ASSIGN TO DISK
ORGANIZATION IS INDEXED
ACCESS MODE IS RANDOM
RECORD KEY IS WERKNEMERS-NUMMER.
```

Het WERKNEMERSBESTAND kan nu worden geopend voor invoer (voor het lezen van een bestaand record met een opgegeven WERK-NEMERS-NUMMER), uitvoer (voor het schrijven van een nieuw record met een opgegeven WERKNEMERS-NUMMER) of invoer-uitvoer (voor het lezen of schrijven zoals hiervoor, of voor rewriting of deleting van een bestaand record met een opgegeven WERKNEMERS-NUMMER).

Tenslotte kan een geïndiceerd bestand gedeeltelijk sequentieel en gedeeltelijk direct worden verwerkt door het specificeren van een toegangstechniek DYNAMIC:

```
SELECT WERKNEMERSBESTAND ASSIGN TO DISK
ORGANIZATION IS INDEXED
ACCESS MODE IS DYNAMIC
RECORD KEY IS WERKNEMERS-NUMMER.
```

In dit geval kunnen alle voorzieningen van de directe toegangstechniek worden toegepast maar bovendien kan het bestand sequentieel worden gelezen vanaf het laatste direct gelezen record of vanaf een bepaald punt, gespecificeerd in een START-opdracht, door een bijzondere vorm van de read-opdracht:

```
READ WERKNEMERSBESTAND NEXT
AT END
.
.
.
```

Men moet inzien dat een bestand eerst moet worden opgebouwd door het wegschrijven (WRITE) van gesorteerde records, waarbij het bestand voor OUTPUT is geopend, voordat een van de mogelijkheden van directe toegang van de index-sequentiële organisatiemethode kan worden gebruikt. Deze allereerste opbouw van het bestand stelt het operating system in staat de nodige index-tabellen op te bouwen. Na deze allereerste opbouw worden de index-tabellen steeds automatisch bijgewerkt.

Opgave

1. Een monteringenbestand bevat records met 100 tekens en is opgebouwd op schijf met geïndiceerde organisatie. De gebruikte record key is MONTERINGS-NR en staat in de eerste 10 posities van het record. MONTERINGS-NR valt in twee delen uiteen: MONTERINGS-GROEP met 5 cijfers, gevolgd door SUB-MONTERING met 5 cijfers. De records in het bestand staan in logische groepen bij elkaar, waarbij iedere groep één kop record voor de gehele montering omvat (nullen staan in het veld SUB-MONTERING van dit record), gevolgd door een variabel aantal sub-monteringsrecords. Deze laatste records hebben alle dezelfde waarde in MONTERINGS-GROEP zoals verschenen in de kop, maar natuurlijk heeft ieder sub-monteringsrecord een uniek getal (binnen de groep) dat niet gelijk is aan nul in het veld SUB-MONTERING.
   Er moet een programma worden geschreven dat de gegevens van alle records voor gegeven monteringen uitlijst. Het nummer van de gewenste montering wordt ingetikt in de eerste 5 posities van een beeldschermregel. Er kunnen meerdere opvragingen in een run voorkomen die niet in een speciale volgorde staan.

Ontwerp de voor dit programma noodzakelijke opdrachten voor File-Control, Record Description en Procedure Division.

T5 WISSELENDE INDICES

Een bijzonder krachtige toepassingsmogelijkheid bij grote compilers is de voorziening: ALTERNATE RECORD KEY. Deze stelt de programmeur in staat te specificeren dat een ander veld in het record, naast de unieke recordsleutel die in de eerste plaats werd gekozen om de logische volgorde van het bestand te regelen, het subject van een index moet zijn die toegang tot de records zal verlenen. Tot een record met een aangegeven alternatieve sleutel kan men dan directe toegang verkrijgen of men kan de records in de sequentiële volgorde van de alternatieve sleutel aflopen. De alternatieve recordsleutel behoeft niet noodzakelijk unieke waarden in de records te hebben, dat wil zeggen dezelfde sleutelwaarde kan in meer dan één record verschijnen.

Het volgende voorbeeld laat zien hoe een VERKOOP-record toegankelijk kan worden gemaakt door middel van VERKOOP-NR, KLANT-NR en VERTEGENW-NR.

```
FILE-CONTROL.
    SELECT VERKOPEN ASSIGN TO DISK
    ORGANIZATION IS INDEXED
    ACCESS MODE SEQUENTIAL
    RECORD KEY IS VERKOOP-NR
    ALTERNATE RECORD KEY IS KLANT-NR WITH DUPLICATES
    ALTERNATE RECORD KEY IS VERTEGENW-NR WITH DUPLICATES.
    .
    .
    .
FD  VERKOPEN.
01  VERKOOP.
    02 VERKOOP-NR      PICTURE 99999.
    02 KLANT-NR        PICTURE 9999.
    02 VERTEGENW-NR    PICTURE 999.
    02 VERKOOP-DETAILS etc.
```

Het bestand wordt in eerste aanleg gemaakt door er records gesorteerd op VERKOOP-NR naar weg te schrijven:

```
OPEN OUTPUT VERKOPEN
repeat until einde-van-invoerbestand
    READ invoerbestand
    AT END
       set einde-van-invoerbestand true
    not end
       WRITE VERKOOPRECORD FROM invoerrecord
       INVALID KEY
          fout afhandelen
```

De enige manier waarop volgens mij een invalid key onder deze omstandigheden kan ontstaan, afgezien van een mankement in de apparatuur, doet zich voor als het wegschrijven van het record tot gevolg zou hebben dat het bestand moest worden uitgebreid boven de maximale bestandstoewijzing die door de programmeur was gedeclareerd.

Is het bestand eenmaal gemaakt, dan kunnen opvolgende programma's door middel van SELECT hiervan naar wens SEQUENTIAL, RANDOM of DYNAMIC gebruik maken. Uitbreiding van het bestand, bijvoorbeeld als gevolg van een WRITE van een nieuw record naar het bestand, wanneer het open stond voor I-O, zal ertoe leiden dat alle drie indices worden bijgewerkt. De enige andere wijziging ten opzichte van de verwerkingen bij de normale geïndiceerde bestandsorganisatie is dat de programma's moeten specificeren welke index moet worden gebruikt, wanneer ze het bestand met READ lezen. Als we bijvoorbeeld de eerste verkoop van klant 27 willen terugvinden:

```
MOVE 27 TO KLANT-NR.
READ VERKOPEN RECORD KEY IS KLANT-NR
INVALID KEY
   PERFORM NIET-ZO'N-KLANT.
```

Het systeem zal er nu verder van uitgaan dat KLANT-NR de sleutel van het record is, bijvoorbeeld in een READ NEXT-instructie, totdat een nadere KEY IS-passage wordt uitgevoerd. Uitgaande van het laatste voorbeeld ziet het gehele verhaal met betrekking tot klantnummer 27 er als volgt uit:

```
repeat until KLANT-NR not = 27 of einde-van-verkopenbestand
   READ VERKOPEN NEXT
   AT END
      set einde-van-verkopenbestand true
   not end
      if KLANT-NR = 27
         verwerk verkooprecord
```

ANTWOORDEN HOOFDSTUK T

Paragraaf T1

1. 200.
2. 10 × 8.000 = 80.000 tekens.
3. 400 × 80.000 = 32 miljoen tekens.
4. Groter. Er zijn evenveel gegevens op ieder spoor terwijl de sporen naar het centrum toe korter zijn dan de buitenste sporen. Daardoor moeten de gegevens op de binnenste sporen met een grotere dichtheid worden vastgelegd.

Paragraaf T4

1.
```
FILE-CONTROL.
    SELECT MONTERINGEN ASSIGN TO DISK
        ORGANIZATION IS INDEXED
        ACCESS MODE IS DYNAMIC
        RECORD KEY IS MONTERINGS-NR.
    SELECT VRAGEN ASSIGN TO CARD-READER.
    SELECT AFDRUKBESTAND ASSIGN TO PRINTER.
:
FD  MONTERINGEN.
01  MONTERINGS-REC.
    02 MONTERINGS-NR.
       03 MONTERINGS-GROEP    PICTURE 9(5).
       03 SUB-MONTERING       PICTURE 9(5).
    02 VERDERE-GEGEVENS       PICTURE X(90).
:
FD  VRAGEN.
:
    02 GEWENSTE-MONTERINGS-GROEP  PICTURE 9(5).
:
PROCEDURE DIVISION.
    OPEN INPUT MONTERINGEN.
    open andere bestanden
    repeat until einde-vragen
        READ VRAGEN RECORD
        AT END
           set einde-vragen true
        not end
           MOVE GEWENSTE-MONTERINGS-GROEP TO MONTERINGS-GROEP
           MOVE ZEROS TO SUB-MONTERING
           READ MONTERINGEN RECORD
           INVALID KEY
               display boodschap "MONTERING NIET GEVONDEN" etc
           not invalid
               repeat until einde-MONTERINGEN or
               MONTERINGS-GROEP not = GEWENSTE-MONTERINGS-GROEP
                  display de inhoud van MONTERINGS-REC
                  READ MONTERINGEN NEXT
                  AT END
                     set einde-MONTERINGEN true.
```

U COMMUNICATIE TUSSEN PROGRAMMA'S

U1 DE BIBLIOTHEEK MET DOELTAAL-PROGRAMMA'S

We zijn het idee van een programmabibliotheek al tegengekomen bij de COPY-opdracht (par. Q1). De COPY-opdracht heeft tot gevolg dat de compiler de code in ons programma tijdens de compilering opneemt door opdrachten in de uitgangstaal (ook wel brontaal genoemd) uit de COBOL-*bibliotheek met uitgangstaalopdrachten* te kopiëren.

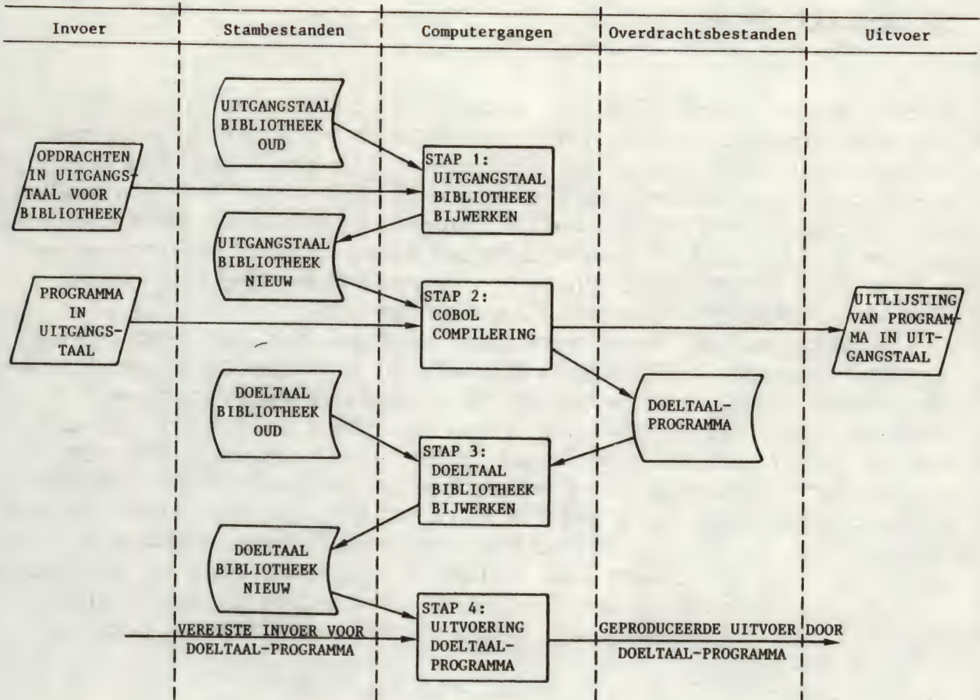
Het is ook mogelijk doeltaal-programma's (Engels: object programs) binnen de machine te brengen tijdens de uitvoering van het programma door ze te kopiëren uit de *bibliotheek met doeltaal-programma's*. Evenals een functie werd gebruikt om opdrachten in de uitgangstaal in de bibliotheek met uitgangstaalopdrachten op te bergen, zo wordt een functie, die vaak wordt opgeroepen door het algemene besturingssysteem van de computer, gebruikt om doeltaal-opdrachten op te bergen in de bibliotheek met doeltaal-programma's.

Deze doeltaal-programma's kunnen het resultaat zijn van een COBOL-compilering of van de compilering van een programma in een andere taal.

In de praktijk bevat gewoonlijk elk doeltaal-programma dat voortvloeit uit een COBOL-compilering vele oproepen van algemene subroutines uit de bibliotheek met doeltaal-programma's. In het algemeen hoeft de programmeur zich niet bewust te zijn van deze oproepen. Typische standaardroutines zijn bijvoorbeeld de omzetting van een decimaal getal in een intern binair getal; het blokken en ontblokken van records; het opsporen van een record op een geïndiceerd bestand bij een gegeven record key. De COBOL-compiler zorgt ervoor dat deze oproepen van de subroutines automatisch tijdens de uitvoering worden gedaan.

De logica van compilering en uitvoering kan nu worden beschreven zoals in figuur 48.

Dit schema suggereert dat het doeltaal-programma tijdens zijn ontwikkeling al wordt toegevoegd aan de bibliotheek met doeltaal-programma's; in de praktijk bestaat de bibliotheek uit een produktie-bibliotheek en een ontwikkelingsbibliotheek. Programma's die nog in ontwikkeling zijn kunnen opgeborgen worden in een van de ontwikkelingsbibliotheken. Pas nadat ze volledig getest en geoperatio-



FIGUUR 48 Compilering en uitvoering met gebruik van de bibliotheken met uitgangstaalopdrachten en doeltaal-programma's.

naliseerd zijn worden ze in de produktiebibliotheek opgeborgen. Gedurende de ontwikkeling kan het door de compiler geproduceerde doeltaal-programma dus ook na fase 2 rechtstreeks in de machine worden geladen bij fase 4, en kan fase 3 worden overgeslagen. Opslag van het programma in een ontwikkelingsgedeelte van de bibliotheek zorgt er anderzijds voor dat het rechtstreeks in de machine kan worden geladen. Daarmee wordt dan de noodzaak om het programma steeds weer te compileren, wanneer het gedraaid wordt, vermeden.

Opgaven

1. (a) Wordt in figuur 48 de bibliotheek met doeltaal-programma's gebruikt in fase 2?
   (b) Denkt u dat het schema helemaal correct is bij fase 2?
   (c) Denkt u dat het schema helemaal correct is bij de fasen 1 en 3?
2. Kunt u een COBOL-programma schrijven zonder iets te weten over de bibliotheek met doeltaal-programma's?

U2 DE CALL-OPDRACHT

COBOL bevat voorzieningen waardoor de programmeur een subroutine kan oproepen die zich in de bibliotheek met doeltaal-programma's bevindt. Dit vereist over het algemeen het opgeven van één of meer parameters, dat wil zeggen variabelen of constanten die door de subroutine moeten worden gebruikt. Een subroutine in de bibliotheek met doeltaal-programma's wordt in het algemeen een *subprogramma* genoemd om het te onderscheiden van een subroutine uitgevoerd door een PERFORM-opdracht in een programma.

In paragraaf K3 hebben we geconstateerd dat het bevolkingsexplosie-vraagstuk (Appendix B) opgevat kan worden als bestaande uit twee duidelijk onderscheiden componenten, waarvan de ene de algehele besturing van het lezen van de invoer en het opleveren van de resultaten verzorgt en de andere het afdrukken van één pagina met resultaten. Deze laatste component had ook als subprogramma geïmplementeerd kunnen worden. Dit subprogramma zou verwachten dat er twee parameters doorgegeven zouden worden bij aanroep, namelijk de grootte van de bevolking bij aanvang en het groeipercentage. Hiermee is het in staat de nieuwe bevolkingsgroottes te berekenen en de regels van een pagina met resultaten te bepalen en af te drukken.

Het algemene formaat van de CALL-opdracht is:

```
CALL "subprogramma-naam" [USING parameter-1 [parameter-2]...]
```

Als we in het programma dat de algehele besturing verzorgt (het hoofdprogramma, of het oproepende (calling) programma) de grootte van de bevolking bij aanvang in een gegevenselement met de naam BEGINBEV en het groeipercentage in een ander gegevenselement met de naam GRPERC hadden geplaatst, zouden we in het hoofdprogramma kunnen coderen:

```
CALL "DRUK-PAGINA-AF" USING BEGINBEV GRPERC.
```

De parameters moeten worden opgegeven in de volgorde die het subprogramma verwacht en ze moeten de pictures hebben die het subprogramma vereist.

Nadat het subprogramma is uitgevoerd, gaat de besturing over naar de op de call volgende COBOL-opdracht. CALL en PERFORM hebben dus een identieke logica; PERFORM werkt met in het programma besloten paragrafen, en CALL met subprogramma's die zich bevinden in de bibliotheek met doeltaal-programma's.

In dit voorbeeld zou het subprogramma onder de naam DRUK-PAGINA-AF in de bibliotheek met doeltaal-programma's opgenomen moeten worden.

U3 HET MET CALL OPGEROEPEN SUBPROGRAMMA

Het met CALL opgeroepen subprogramma kan worden geschreven in iedere taal die de computerfabrikant voor dit doel toestaat. De fabrikant stelt *richtlijnen* vast voor het schrijven van deze programma's, zodat later een geslaagde oproep kan worden gedaan. Zijn zowel het oproepende als het opgeroepen programma in COBOL geschreven, dan worden deze richtlijnen automatisch verzorgd en hoeft de programmeur er zich niet om te bekommeren.

Een COBOL-subprogramma verschilt in twee opzichten van een normaal COBOL-programma. Deze betreffen de *acceptatie van de parameters* en de *terugkeer naar het oproepende programma.*

Acceptatie van de parameters. Wanneer het subprogramma aan de hand van een *naam (call by reference)* wordt opgeroepen, werkt dit subprogramma met (parameter)gegevens die zich in velden binnen het oproepende programma bevinden. Wanneer een subprogramma aan de hand van een *waarde (call by content)* wordt opgeroepen, worden de gegevens verplaatst van de velden in het oproepende programma naar de velden in het opgeroepen programma. Het enige praktische verschil is dat bij het oproepen met een waarde het subprogramma de inhoud van zijn parametervelden kan aanpassen (of verknoeien) zonder de parametervelden in het oproepende programma te wijzigen.

Standaard COBOL-programma's worden uitgevoerd op basis van een oproep met een naam, hoewel CODASYL heeft voorgesteld de mogelijkheid van oproep met een waarde in te voeren. In beide gevallen bevinden de gegevens zich oorspronkelijk buiten het subprogramma. Om deze reden staan de parametervelden bekend als *externe* variabelen van het opgeroepen subprogramma.

Deze externe variabelen moeten op de een of andere wijze worden aangeduid. COBOL stelt als verplichting dat externe variabelen bij elkaar worden gegroepeerd in een speciale section van de Data Division van het subprogramma, de LINKAGE SECTION. In dat geval wordt een bijzondere declaratievorm van de Procedure Division gebruikt:

```
PROCEDURE DIVISION [USING parameter-1 [parameter-2]...].
```

waar de parameters de namen van de externe variabelen zijn. De als parameters gebruikte gegevensnamen behoeven niet dezelfde te zijn als die in het oproepende programma worden gebruikt, maar ze moeten wel in dezelfde volgorde verschijnen en dezelfde picture hebben.

Terugkeer naar het oproepende programma wordt bereikt door het coderen van EXIT PROGRAM aan het einde van het subprogramma. De besturing keert terug naar de op CALL volgende opdracht in het hoofdprogramma.

Ons voorbeeld uit paragraaf U2 voortzettend zou het subprogramma als dit in COBOL werd geschreven, er als volgt uit kunnen zien:

```
:
DATA DIVISION.
:
LINKAGE SECTION.
01  PARAMETERS.
    02 BEGINWAARDE-BEV  PICTURE 9(10).
    02 GROEI-FACTOR     PICTURE 9(6)V9(9).
PROCEDURE DIVISION USING BEGINWAARDE-BEV GROEI-FACTOR.
:
    MOVE BEGINWAARDE-BEV TO W-BEV
    Repeat voor elke decade
       COMPUTE W-BEV ROUNDED = W-BEV*GROEI-FACTOR
       ON SIZE ERROR
          druk size error regel af
          EXIT PROGRAM
       not size error
          bereken dichtheid
          on size error
             druk size error regel af
             EXIT PROGRAM
          not size error
             druk regel met resultaten af
    EXIT PROGRAM.
```

Merk op hoe het gebruik van EXIT PROGRAM in dit voorbeeld een uit de herhaling springende GOTO overbodig maakt. CODASYL heeft voorgesteld een soortgelijke opdracht, EXIT PERFORM, in te voeren voor gebruik bij interne subroutines.

In dit voorbeeld is niet nader ingegaan op het probleem hoe in- en uitvoer-operaties plaatsvinden in het subprogramma, wanneer zowel het opgeroepen als het oproepende programma van dezelfde bestanden gebruik moeten maken. Door verschillende computerfabrikanten wordt dit probleem op verschillende wijze opgelost. Sommigen staan een globale toegang tot de bestanden toe; anderen bieden de mogelijkheid om een verwijzing naar een bestand in de lijst van parameters op te geven. Raadpleeg het technische handboek.

Zoals reeds gezegd in deel 2 is het gebruik van een apart gecompileerd subprogramma voor de oplossing van het bevolkingsexplosievraagstuk nogal overdreven. Er is een nogal grote hoeveelheid extra codering vereist bij het gebruik van subprogramma's in COBOL en dat is een belemmering om ze veelvuldig toe te passen voor kleine routines, die daarom gewoonlijk beter afgehandeld kunnen worden met behulp van interne subroutines. Toch wordt een methode van werken aanbevolen waarbij de programmeur duidelijk samenhangende

functies onderbrengt in subprogramma's met bescheiden omvang, elk met eigen lokale variabelen en expliciete parameters en elk met de mogelijkheid separaat getest te worden.

Opgaven

1. In de tekst werd gesuggereerd dat de als parameters gebruikte variabelen dezelfde picture in het opgeroepen en oproepende programma moeten hebben. Dit is niet helemaal waar. Kunt u opgeven in welke belangrijke opzichten de pictures gelijkluidend moeten zijn?

2 U ontwerpt een subprogramma dat werkt op 100 velden in het oproepende programma. Waarschijnlijk is hier een fout gemaakt (het bereik van de handelingen die in het subprogramma zijn opgenomen is te groot), maar aannemend dat u hier toch mee wilt doorgaan, hoe kunt u dan vermijden dat bij elke aanroep 100 namen als parameters worden opgegeven?

U4 SEGEMENTERING EN OVERLAY

Opdrachten in de Procedure Division kunnen worden verdeeld in sections. We bereiken dit door te declareren:

```
section-name SECTION [priority-number].
```

De door de compiler geproduceerde doelcode voor een section wordt een segment genoemd. Ieder segment kan afzonderlijk worden opgeborgen onder zijn section-name in de bibliotheek met doeltaal-programma's.

Deze segmentering is waardevol wanneer de voor het gehele programma geproduceerde doelcode meer plaats in het geheugen vereist dan beschikbaar is in de machine die wordt gebruikt, dat wil zeggen wanneer het programma zo groot is dat het niet past in de computer. Op grote moderne computers levert dit geen problemen meer op, daar het besturingssysteem een groot programma automatisch in segmenten zal verdelen zonder dat de operator daar iets voor hoeft te doen. Op kleinere machines met minder geavanceerde besturingssystemen kunnen er wel problemen optreden. Zelfs op een grote machine kan het gewenst zijn controle uit te oefenen over de segmentering om de verwerkingstijd van een programma te verminderen. De programmeur kan dan zorgen voor een aantal programmasegmenten die alleen in de machine worden opgeroepen wanneer ze nodig zijn en die dezelfde opslagplaatsen bezetten (*overlaying*) als een eerder uitgevoerd segment.

Een voorbeeld kan dit duidelijk maken. Veronderstel dat een zeer groot programma moet worden geschreven om een overzicht te produceren. Het programma verloopt in drie fasen. In de eerste fase worden besturingsrecords gelezen die worden gebruikt om tabellen en indicatoren te vormen die de inhoud en de vorm van het rapport bepalen. In de tweede fase wordt een bestand in zijn geheel gelezen en worden totalen bijgehouden. Tenslotte wordt in de derde fase het overzicht geproduceerd onder gebruikmaking van deze berekende totalen.

Om het programma in de machine te laten passen veronderstellen we dat de codering voor iedere fase een afzonderlijk segment vormt, hetgeen drie segmenten oplevert. De Procedure Division kan dan als volgt worden gecodeerd:

```
PROCEDURE DIVISION.
PERMANENT SECTION 0.
A.
    PERFORM BESTURINGSRECORD.
    PERFORM TE-VERWERKEN-BESTAND.
    PERFORM GEEF-OVERZICHT.
    STOP RUN.
BESTURINGSRECORD SECTION 2.
    (codering om de besturingskaarten in te lezen, etc.)
TE-VERWERKEN-BESTAND SECTION 1.
    (codering om het bestand te lezen en totalen bij te houden)
GEEF-OVERZICHT SECTION 3.
    (codering om het overzicht te produceren)
```

De prioriteitsnummers zullen dadelijk worden toegelicht; ze worden alleen gebruikt om de overlay-sections te onderscheiden van de permanent aanwezige sections.

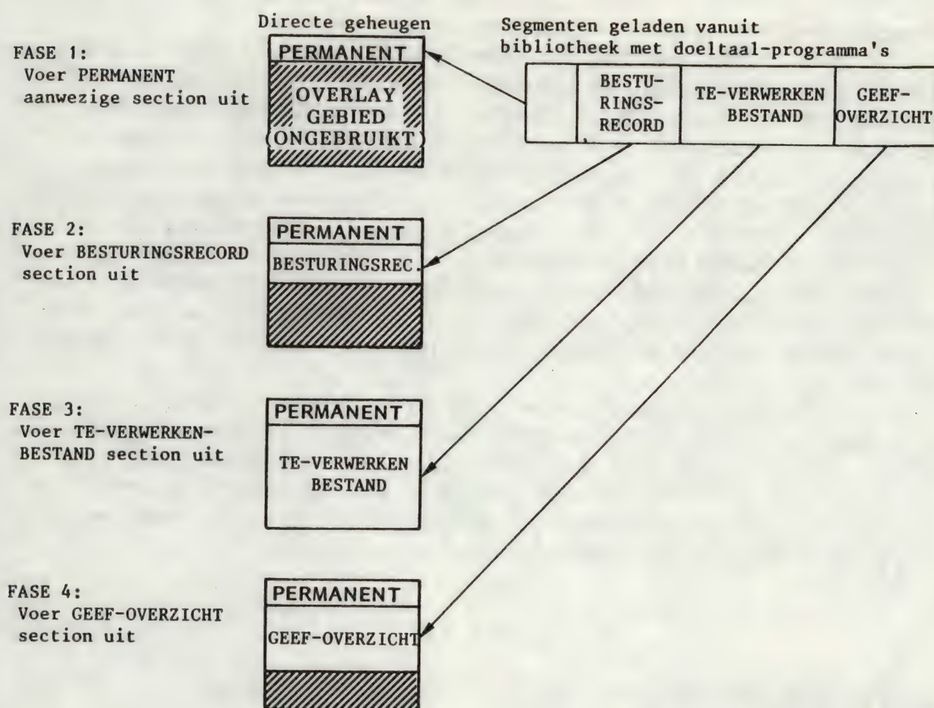
De volgorde van handelingen in de uitvoering van dit programma kan in het diagram op de volgende pagina zichtbaar worden gemaakt.

We zien dat de geheugenruimte die benodigd is voor de procedures van dit programma gelijk is aan de som van de permanent aanwezige section plus de grootste overlay-section. Het overbrengen van de segmenten naar het directe geheugen wordt verzorgd door het besturingssysteem zodra ze benodigd zijn.

Het overbrengen van een segment neemt natuurlijk enige tijd in beslag. Indien een segment herhaaldelijk wordt gebruikt, bijvoorbeeld in een lus, kan de tijd die benodigd is voor invoer van de overlays langer zijn dan de tijd om het segment te verwerken.

U vraagt zich op dit ogenblik wellicht af: "Hoe weet ik van tevoren dat mijn programma zo groot is dat het niet in de machine past? Indien ik van het ongunstigste uitga en mijn programma drastisch segmenteer, kan ik wel eens uitkomen op betaling van een onnodige tijdboete wegens overmatige overlay-activiteit."

Voor de oplossing van dit probleem bestaan de prioriteitsnummers. Aan iedere section wordt een prioriteitsnummer van 0 t/m 49 toegewezen. Aan vaak gebruikte sections wordt een laag nummer



FIGUUR 49 Fasen in een programma met overlays.

gegeven (hetgeen hoge prioriteit aangeeft), terwijl zelden gebruikte sections een hoog nummer krijgen, dat wil zeggen toenemende prioriteitsnummers worden toegewezen aan sections naarmate hun frequentie in het gebruik afneemt. Een section die zo vaak wordt gebruikt dat hij permanent aanwezig moet zijn (of een section die te klein is om je druk over te maken zoals in dit voorbeeld) krijgt een prioriteitsnummer 0. Twee of meer sections met hetzelfde prioriteitsnummer kunnen in de bibliotheek met doeltaal-programma's samengevoegd worden om één segment van doelcode te vormen.

Een segmentgrensnummer wordt gedefinieerd in de paragraaf Object-computer. Sections met een lager prioriteitsnummer dan het segmentgrensnummer zijn altijd permanent aanwezig. Sections met een prioriteitsnummer dat gelijk is aan of hoger dan het segmentgrensnummer worden in overlay verwerkt.

Door dit middel kan bij de eerste test van het programma een hoog segmentgrensnummer worden gekozen, bijvoorbeeld 49. Dit heeft tot gevolg dat alle sections permanent aanwezig zijn. Als het programma in het geheugen past, gaat alles goed. Zo niet, dan kan het programma opnieuw worden gecompileerd met een lager segmentgrensnummer, zodat de minst gebruikte sections in overlay worden

verwerkt. Dit proces wordt herhaald totdat het gesegmenteerde programma zo klein is dat het past in de machine.

In de praktijk bestaat een soortgelijke voorziening van overlay voor opgeroepen subprogramma's, indien de compiler deze heeft geïmplementeerd als 'Co-routine's'. Als een subprogramma een 'co-routine' is, wordt het niet permanent aan het hoofdprogramma gekoppeld, maar blijft apart staan in de bibliotheek met doeltaal-programma's en wordt alleen tijdens de verwerking bij aanroep door een CALL-opdracht in het geheugen gehaald. Normaliter zal het, nadat het eenmaal is opgeroepen, in het geheugen blijven staan, waarbij zijn variabelen van de ene CALL tot de volgende onveranderd zullen blijven. Maar indien de programmeur dat wenst, kan hij de geheugenruimte die door het subprogramma in beslag wordt genomen, vrijgeven door een CANCEL-opdracht te geven: CANCAL "subprogramma-naam". De vrijgegeven geheugenruimte kan dan voor andere subprogramma's (of voor andere programma's bij een multi-programmeringscomputer) benut worden. Als nu in een volgende CALL-opdracht hetzelfde subprogramma wordt opgeroepen zal het opnieuw geladen worden en krijgen de variabelen de waarden zoals die bij compilatie zijn gedeclareerd.

ANTWOORDEN HOOFDSTUK U

Paragraaf U1

1. (a) Neen.
   (b) Neen. Alle programma's van enige complexiteit sluiten waarschijnlijke oproepen naar subroutines in die in de bibliotheek met doeltaal-programma's staan en de compiler is zeker een ingewikkeld programma.
   (c) Neen.
2. Hebt u dat al niet gedaan?

Paragraaf U3

1. Ze moeten dezelfde lengte hebben. Numerieke elementen die moeten worden gebruikt bij rekenkundige bewerkingen moeten dezelfde USAGE-passage hebben en de decimale punt op dezelfde plaats.
2. Plaats alle 100 variabelen tezamen in een groepsveld en geef het groepsveld als de parameter op. (Opmerking: wanneer we het aantal parameters dat bij de aanroep van een subroutine moet worden opgegeven beschouwen vanuit het criterium van de reikwijdte van onze aandacht, dan telt een groepsveld dat een duidelijke gegevensstructuur weerspiegeld, zoals een record of tabel, slechts als één parameter.)

APPENDIX A
VOORBEELDPROGRAMMA

DE PROGRAMMASPECIFICATIE

Doel

De gesorteerde geldige mutatierecords moeten gebruikt worden om het klantenbestand bij te werken door een nieuw klantenstambestand op te bouwen. Fouten die tijdens de verwerking worden ontdekt moeten op de regeldrukker worden afgedrukt.

Invoer

Het klantenstambestand (identificatie: KLANTEN) bestaat uit records met de volgende indeling:

1-6	klant-nummer 9(6)
7	krediet-code X
8-9	gebieds-code XX
10-169	naam en adres van klant (4 regels van elk 40 tekens)

Het mutatiebestand (identificatie: MUT) heeft records met dezelfde indeling als KLANTEN, behalve dat er een additioneel teken aan het begin van elk record staat dat de mutatiecode bevat. Dit kan zijn:

I	voor invoegen
W	voor wijzigen en
V	voor verwijderen.

Uitvoer

Het klantenstambestand dat wordt uitgevoerd is identiek aan het klantenstambestand dat wordt ingevoerd (in de bestandsbibliotheek moeten de bestanden van elkaar onderscheiden worden door een generatie- of versienummer dat door het besturingssysteem wordt toegekend).

Het foutenverslag moet van geschikte kopregels worden voorzien en bestaat uit de volgende detailregels:

1-6	klant-nummer
10	krediet-code
15-16	gebieds-code
19-58	naam en adres (voort te zetten over 4 regels)

62 mutatie-code
66- foutboodschap, te weten één van de twee volgende:
1. "INVOEGING-RECORD ZIT AL IN STAM"
2. "GEEN STAMRECORD VOOR DEZE KLANT".

Verwerking

KLANTEN en MUT zijn beide in opklimmende volgorde van klantnummer gesorteerd. De waarden van de velden van MUT zijn reeds gevalideerd en records met eenzelfde klantnummer zijn volgens mutatiecode in de volgorde I, W en V gesorteerd.

Aan de hand van KLANTEN en MUT moet een bijgewerkt nieuw klantenbestand worden opgebouwd. Als de mutatiecode een invoeging aangeeft moet het mutatierecord (zonder de mutatiecode) op de juiste plaats in het nieuwe klantenbestand worden geplaatst, als tenminste niet al een stamrecord voor deze klant bestaat. Als er al wel een klant met dit klantnummer bestaat moet het mutatierecord afgedrukt worden met bericht 1. Een invoeging wordt nooit gevolgd door een wijziging of verwijdering van diezelfde invoeging.

Als de mutatiecode in MUT een wijziging aangeeft, moet elk nietleeg veld van het mutatierecord het overeenkomstige veld in het klantenrecord vervangen. Een wijzigingsrecord kan worden gevolgd door verdere wijzigingsrecords of een verwijdering voor hetzelfde klantenrecord. Als geen verwijdering volgt moet het klantenrecord, nadat alle wijzigingen zijn aangebracht, naar het nieuwe klantenbestand worden overgebracht.

Als de mutatiecode een verwijdering aangeeft moet het klantenrecord niet worden overgebracht naar het nieuwe klantenbestand. Na een verwijdering volgt nooit meer een ander mutatierecord voor dezelfde klant.

Als er bij een wijziging of verwijdering geen klantenrecord bestaat met hetzelfde klantnummer moet dit mutatierecord worden afgedrukt met bericht 2.

Tussen elk tweetal afgedrukte records van het foutenverslag en ook tussen de kop en het eerste afgedrukte record moeten twee blanco regels worden opgenomen. Er mogen niet meer dan 50 regels op een pagina staan, behalve wanneer daardoor de gegevens van een en hetzelfde record over twee pagina's gesplitst zouden moeten worden. In dat geval mag het aantal van 50 zoveel als nodig is overschreden worden.

DE PROGRAMMA-UITLIJSTING

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  KSBB1.
*AUTHOR.   A.PARKIN.
*DATE-WRITTEN.  SEPTEMBER 1983.
*
*BIJWERKEN KLANTENSTAMBESTAND
*============================
```

```
*LEES MUTATIEBESTAND
*LEES OUDE STAMBESTAND
*HERHAAL TOTDAT EINDE-MUTATIEBESTAND
*   HERHAAL TOTDAT EINDE-OUDE-STAMBESTAND
*   OF STAM-RECORD-SLEUTEL NIET KLEINER DAN
*   MUTATIE-RECORD-SLEUTEL
*      BRENG OUDE STAMBESTAND RECORD NAAR
*         NIEUW STAMBESTAND RECORD EN SCHRIJF DIT WEG
*      LEES OUDE STAMBESTAND
*   VERGELIJK STAM-RECORD-SLEUTEL MET MUTATIE-SLEUTEL
*   EN EVALUEER MUTATIE-CODE
*   WANNEER = EN WIJZIGING
*      DOE 'WIJZIGEN OUDE STAMBESTAND RECORD'
*   WANNEER = EN VERWIJDERING
*      DOE LEES VOLGENDE OUDE STAMBESTAND RECORD
*   WANNEER = EN INVOEGING
*      FOUT AFDRUKKEN-INVOEGING BESTAAND RECORD
*   WANNEER (EINDE-OUDE-STAMBESTAND OF NIET =)
*                                  EN INVOEGING
*      BRENG MUTATIERECORD MINUS MUTATIE-CODE
*            OVER NAAR NIEUW STAMBESTAND RECORD
*            EN SCHRIJF DIT WEG
*   WANNEER (EINDE-OUDE-STAMBESTAND OF NOT =)
*               EN (WIJZIGING OF VERWIJDERING)
*      FOUT AFDRUKKEN-GEEN STAM
*HERHAAL TOTDAT EINDE-OUDE-STAMBESTAND
*   BRENG OUDE STAMBESTAND RECORD NAAR
*      NIEUWE STAMBESTAND RECORD EN SCHRIJF DIT WEG
*   LEES OUDE STAMBESTAND
*
*WIJZIGEN OUDE STAMBESTAND RECORD
*--------------------------------
*VOOR IEDER NIET-BLANCO VELD IN MUTATIERECORD
*   BRENG DIT VELD OVER NAAR OVEREENKOMSTIGE
*   VELD IN OUDE STAMBESTAND RECORD
*
*N.B. EEN WIJZIGING KAN WORDEN GEVOLGD DOOR NOG MEER
*WIJZIGINGEN VOOR DEZELFDE KLANT (GELDT NIET VOOR
*EEN VERWIJDERING).
*
*PROGRAMMA ORGANISATIE
*====================
*A1-BIJWERKEN-KLANTENSTAMBESTAND
*   B1-VERWERK-MUTATIE
*      C1 = B2-SCHRIJF-EN-LEES-STAM
*      C2-BIJWERKEN-OUDE-STAM
*   B2-SCHRIJF-EN-LEES-STAM
/
```

```
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT FA-OUD-STAMBESTAND   ASSIGN TO DISK.
    SELECT FB-MUT               ASSIGN TO DISK.
    SELECT FC-NIEUW-STAMBESTAND ASSIGN TO DISK.
    SELECT FD-AFDRUKBESTAND     ASSIGN TO PRINTER.
*            RAADPLEEG UW HANDBOEK VOOR DE JUISTE NAAM
*            VOOR REGELDRUKKER EN SCHIJVENPAKKET OP UW
*            APPARATUUR

 DATA DIVISION.
*
*
 FILE SECTION.
*
 FD  FA-OUD-STAMBESTAND.
 01  FA-OUD-STAMRECORD.
     02 FA-KLANT-NUMMER   PICTURE 9(6).
     02 FA-KREDIET-CODE   PICTURE X.
     02 FA-GEBIEDS-CODE   PICTURE XX.
     02 FA-NAAM-ADRES     PICTURE X(40).
*       VIER REGELS VAN ELK 10 POSITIES BIJ
*       ALLEREERSTE TESTEN
*
 FD  FB-MUT.
 01  FB-MUTRECORD.
     02 FB-MUTATIE-CODE   PICTURE X.
        88 INVOEGING      VALUE IS "I".
        88 WIJZIGING      VALUE IS "W".
        88 VERWIJDERING   VALUE IS "V".
     02 FB-MUT-DETAILS.
        03 FB-KLANT-NUMMER PICTURE 9(6).
        03 FB-KREDIET-CODE PICTURE X.
        03 FB-GEBIEDS-CODE PICTURE XX.
        03 FB-NAAM-ADRES.
           04 FB-NAW-REGEL OCCURS 4 TIMES PICTURE X(10).
*
 FD  FC-NIEUW-STAMBESTAND.
 01  FC-NIEUW-STAMRECORD  PICTURE X(49).
*
 FD  FD-AFDRUKBESTAND REPORT IS R-FOUTEN.

 WORKING-STORAGE SECTION.
*
 01  WA-FOUTMELDINGEN.
*    EEN VAN BEIDE MELDINGEN WORDT GEBRUIKT IN EEN
*    R-FOUTEN-GROEP
     02 FILLER            PICTURE X(33)
     VALUE "INVOEGING - RECORD ZIT AL IN STAM".
```

```
        02 FILLER              PICTURE X(33)
           VALUE "GEEN STAMRECORD VOOR DEZE KLANT".
    01  WA-MELDINGEN REDEFINES WA-FOUTMELDINGEN.
        02 WA-MELDING          PICTURE X(33)
           OCCURS 2 TIMES INDEXED BY WA.
   *
    01  WB-GLOBALE-CONDITIES.
        02 WB-EVB-OUD-STAMBESTAND-TEKEN  PICTURE X.
           88 EVB-OUD-STAMBESTAND VALUE IS "W".
        02 WB EVB-MUT-TEKEN              PICTURE X.
           88 EVB-MUT             VALUE IS "W"
   /
```

```
REPORT SECTION.
RD  R-FOUTEN PAGE LIMIT 20 LINES
*            20 REGELS BIJ EERSTE TESTS
01  TYPE PAGE HEADING.
    02 LINE 2.
       03 COLUMN 14    PICTURE X(44)
          VALUE " BIJWERKEN KLANTENSTAMBESTAND - FOUTENVERSLAG".
       03 COLUMN 65    PICTURE X(6)
          VALUE "PAGINA".
       03 COLUMN 72    PICTURE Z9
          SOURCE PAGE-COUNTER.
    02 LINE PLUS 2.
       03 COLUMN 1     PICTURE X(12)
          VALUE "KLANTNR KRDT".
       03 COLUMN 15    PICTURE X(11)
          VALUE "GEBIED  NAW".
       03 COLUMN 35    PICTURE X(11)
          VALUE "CODE".
*
01  R-FOUTEN-GROEP TYPE DETAIL.
    02 LINE PLUS 2.
       03 COLUMN 1     PICTURE 9(6)
          SOURCE FB-KLANT-NUMMER.
       03 COLUMN 11    PICTURE X
          SOURCE FB-KREDIET-CODE.
       03 COLUMN 17    PICTURE XX
          SOURCE FB-GEBIEDS-CODE.
       03 COLUMN 23    PICTURE X(10)
          SOURCE FB-NAW-REGEL(1).
       03 COLUMN 37    PICTURE X
          SOURCE FB-MUTATIE-CODE.
       03 COLUMN 40    PICTURE X(33)
          SOURCE WA-MELDING(WA).
    02 LINE PLUS 1.
       03 COLUMN 23    PICTURE X(10)
          SOURCE FB-NAW-REGEL(2).
    02 LINE PLUS 1.
       03 COLUMN 23    PICTURE X(10)
          SOURCE FB-NAW-REGEL(3).
    02 LINE PLUS 1.
       03 COLUMN 23    PICTURE X(10)
          SOURCE FB-NAW-REGEL(4).
*
01  LINE PLUS 2 TYPE REPORT FOOTING.
    02 COLUMN 1          PICTURE X(20)
       VALUE "EINDE FOUTEN-VERSLAG".
```

```
 PROCEDURE DIVISION.
*
 A1-BIJWERKEN-KLANTENSTAMBESTAND.
     OPEN INPUT FB-MUT.
     MOVE "O" TO WB-EVB-MUT-TEKEN.
     READ FB-MUT
     AT END
        DISPLAY "PROGRAMMA-FOUT A1 - LEEG MUTATIEBESTAND"
        DISPLAY "BIJWERKEN KLANTENSTAMBESTAND BEEINDIGD"
        CLOSE FB-MUT
        STOP RUN.
     OPEN INPUT FA-OUD-STAMBESTAND.
     MOVE "O" TO WB-EVB-OUD-STAMBESTAND-TEKEN.
     READ FA-OUD-STAMBESTAND
     AT END
        MOVE "W" TO WB-EVB-OUD-STAMBESTAND-TEKEN.
     OPEN OUTPUT FD-AFDRUKBESTAND.
     INITIATE R-FOUTEN.
     OPEN OUTPUT FC-NIEUW-STAMBESTAND.
     PERFORM B1-VERWERK-MUTATIE UNTIL EVB-MUT.
     CLOSE FB-MUT.
     PERFORM B2-SCHRIJF-EN-LEES-STAM UNTIL EVB-OUD-STAMBESTAND.
     CLOSE FA-OUD-STAMBESTAND FC-NIEUW-STAMBESTAND WITH LOCK.
     TERMINATE R-FOUTEN.
     CLOSE FD-AFDRUKBESTAND.
     STOP RUN.
*
 B1-VERWERK-MUTATIE.
     PERFORM B2-SCHRIJF-EN-LEES-STAM UNTIL EVB-OUD-STAMBESTAND
     OR FA-KLANT-NUMMER NOT LESS THAN FB-KLANT-NUMMER.
     IF (NOT EVB-OUD-STAMBESTAND) AND
        FA-KLANT-NUMMER = FB-KLANT-NUMMER
        IF WIJZIGING
           PERFORM C2-BIJWERKEN-OUDE-STAM
        ELSE
           IF VERWIJDERING
              READ FA-OUD-STAMBESTAND
              AT END
                 MOVE "O" TO WB-EVB-OUD-STAMBESTAND-TEKEN
           ELSE
              SET WA TO 1
              GENERATE R-FOUTEN-GROEP
     ELSE
*    DUS OF EVB-OUD-STAMBESTAND OF FA-KLANT-NUMMER >
*    FB-KLANT-NUMMER
        IF INVOEGING
           WRITE FC-NIEUW-STAMRECORD FROM FB-MUT-DETAILS
        ELSE
*    MOET EEN WIJZIGING OF VERWIJDERING ZIJN
        SET WA TO 2
        GENERATE R-FOUTEN-GROEP.
```

```
       READ FB-MUT
       AT END
          MOVE "W" TO WB-EVB-MUT-TEKEN.
  *
   B2-SCHRIJF-EN-LEES-STAM.
       WRITE FC-NIEUW-STAMRECORD FROM FA-OUD-STAMRECORD.
       READ FA-OUD-STAMBESTAND
       AT END
          MOVE "W" TO WB-EVB-OUD-STAMBESTAND-TEKEN.
  *
   C2-BIJWERKEN-OUDE-STAM.
       IF FB-KREDIET-CODE NOT = SPACE
          MOVE FB-KREDIET-CODE TO FA-KREDIET-CODE.
       IF FB-GEBIEDS-CODE NOT = SPACES
          MOVE FB-GEBIEDS-CODE TO FA-GEBIEDS-CODE
       IF FB-NAAM-ADRES NOT = SPACES
          MOVE FB-NAAM-ADRES TO FA-NAAM-ADRES.
```

VOORBEELD VAN TESTGEGEVENS

TEST 1

Klantenstambestand (1e versie)

```
000001AAAF. MULDER KERKSTR 5 7522 JG   ZWOLLE
000002AA2J. NOVI   NIEUWSTR 67525 PK   MAARSSEN
000010ZZAA. KORFF  HERTOGLN 72523 AN   DEN HAAG
999998ZZZJ. STOMPS WOERTST 162597 PK   DEN HAAG
```

Mutatiebestand

```
I000001BBBO. BOMMEL PARKLN 8  9999 ZZ   ROMMELDAM1)
W000002B                                          2)
W000002 XX                                        2)
W000002   NIEUWE-NM NIEUW ADR NIEUW ADR NIEUW ADR2)
V000010                                           3)
I000500YYYA.HENDRIKSH.D.MANPK73411 ZN   LOPIK     4)
W000600CCCN.V.VEEN  DAMSIGT 392424 PS   VOORBURG  5)
V000700VVVNOBODY    NOWHERE   0000 AA   NOTHING   6)
I999999HHHI.GEERLINGVIERSL. 632317 KL   BLARICUM  7)
```

Het mutatiebestand eindigt na het stambestand.

Voorspellingen bij mutaties:

1) Deze moet fout gaan: klantnummer bestaat al.
2) Drie mutaties voor dezelfde klant die uitgevoerd zullen worden.
3) Een verwijdering van klant 10 die uitgevoerd zal worden.
4) Een correcte invoeging.
5) en 6) Een wijziging en een verwijdering waarvoor geen stamrecords bestaan. Beide moeten foutmeldingen opleveren.
7) Een correcte invoeging aan het einde van het stambestand.

Het foutenverslag zal er als volgt uitzien:

```
          BIJWERKEN KLANTENSTAMBESTAND FOUTENVERSLAG
KLANTNR KRDT  GEBIED  NAW        CODE
000001   B     BB     O. BOMMEL    I  INVOEGING - RECORD ZIT AL IN STAM
                      PARKLN 8
                      9999 ZZ
                      ROMMELDAM

000600   C     CC     N.V.VEEN     W  GEEN STAMRECORD VOOR DEZE KLANT
                      DAMSIGT 39
                      2424 PS
                      VOORBURG

000700   V     VV     NOBODY       V  GEEN STAMRECORD VOOR DEZE KLANT
                      NOWHERE
                      0000 AA
                      NOTHING
```

(en op de volgende pagina)

```
EINDE FOUTEN-VERSLAG
```

Klantenstambestand (2e versie) zal er als volgt uitzien:

```
000001AAAF. MULDER KERKSTR 5 7522 JG   ZWOLLE     correct
000002BXXNIEUWE NM NIEUW ADR NIEUW ADR NIEUW ADR  gewijzigd
000500YYYA.HENDRIKSH.D.MANPK73411 ZN   LOPIK      ingevoegd
999998ZZZJ.STOMPS  WOERTST 162597 PK   DEN HAAG   correct
999999HHHI.GEERLINGVIERSL. 632361 KL   BLARICUM   ingevoegd
```

(Record van klant 10 is terecht verwijderd)

TEST 2

Klantenstambestand (2e versie)

Mutatiebestand

```
I000000IIIH. BOTH   WOERTSTR162597 PK   DEN HAAG1)
I999997MMMP. SCHALK RUYTERLN3 2020 PP   LISSE   2)
```

1) Een invoeging helemaal aan het begin van het bestand.
2) Een invoeging voordat het einde is bereikt.

Het mutatiebestand eindigt nu eerder dan het stambestand.

Het klantenbestand (3e versie) zal luiden:

```
000000IIIH. BOTH   WOERTSTR162597 PK   DEN HAAG   ingevoegd
000001AAAF. MULDER KERKSTR 5 7522 JG   ZWOLLE     correct
000002BXXNIEUWE NM NIEUW ADR NIEUW ADR NIEUW ADR  correct
000500YYYA.HENDRIKSH.D.MANPK73411 ZN   LOPIK      correct
999997MMNP. SCHALK RUYTERLN3 2020 PP   LISSE      ingevoegd
999998ZZZJ.STOMPS  WOERTSTR162597 PK   DEN HAAG   correct
999999HHHI.GEERLINGVIERSL. 632361 KL   BLARICUM   correct
```

APPENDIX B EEN BLOEMLEZING VAN PRAKTIJKOPGAVEN

1 De bevolkingsexplosie

De wereldbevolking in zeker jaar omvat 4.000 miljoen zielen. De aarde is een bol met een straal van 6.350 km en tweederde van haar oppervlak is water. Het oppervlak van een bol is:
4 × 3.1416 × het kwadraat van de straal.

Schrijf een programma dat de wereldbevolking en de bevolkingsdichtheid (aantal bewoners per vierkante km) met intervallen van 10 jaar voor de volgende 20 decaden afdrukt (waarbij de eerste detailregel de bevolking na afloop van de eerste decade weergeeft). De afgedrukte tekst moet er als volgt uitzien:

Eerste regel	- Posities	1-5	Aangenomen groeipercentage % (Z9.99)
Opvolgende regels	- Posities	1-4	Het jaartal (vier cijfers)
		8-17	De bevolking (99,999,999) in miljoenen
		23-29	De dichtheid/vierk.km (999,999).

Tussen de regels moet een dubbele interlinie worden aangehouden en niet-significante nullen in de antwoorden moeten worden weggelaten.

Het groeipercentage is aangebracht op een record in de kolommen 1-4 met een denkbeeldige decimale punt tussen het tweede en derde cijfer. Er kan meer dan één record zijn, in welk geval de resultaten met betrekking tot ieder aangenomen groeipercentage moeten worden afgedrukt op een nieuwe pagina (een pagina per ingelezen record).

Als een SIZE ERROR optreedt in de berekeningen voor een record, druk dan SIZE ERROR af in de posities 1 t/m 10 en ga door met het volgende record.

2 Aanwezigheid van studenten

Een studentenbestand is opgeslagen in de volgende vorm:

```
Identificatie: STUDENTEN
Gegevensrecords (in blokken van tien)
     1-6   Studentnummer
     7-25  Studentnaam
             7-21  Familienaam
            22-25  Beginletters van voornamen
    26-30  Cursuscode
    31-33  Totaal aantal uren gegeven
    34-36  Totaal aantal uren gevolgd
    37-39  Maximale score voor practicum
    40-42  Behaalde score voor practicum
```

Er moet een overzicht worden gemaakt dat de namen van die studenten noemt die 25% of meer lesuren hebben gemist en daardoor gevaar lopen zich te diskwalificeren voor het eindexamen. Het overzicht dient als volgt te worden afgedrukt:

```
 1-6   Studentnummer
10-28  Studentnaam
35-39  Cursuscode
45-47  Totaal aantal uren gegeven
53-55  Totaal aantal uren gevolgd
61-62  % aanwezigheid (twee cijfers; rest afkappen;
                        gevolgd door percentage-teken).
```

3 Bevolkingsexplosie: deel II

Wijzig het overzicht dat in opgave 1 is gemaakt als volgt:
(a) Een kop BEVOLKINGSRAMING: AANGENOMEN GROEIPERCENTAGE = Z9.99% moet worden afgedrukt boven aan iedere pagina, gevolgd door een kop JAAR BEVOLKING DICHTH/VIERK.KM. die TWEE MAAL over de pagina wordt afgedrukt, dat wil zeggen te beginnen bij positie 1 en opnieuw bij positie 60.
(b) De resultaten moeten worden uitgelijst in de aldus gevormde paginahelften: de eerste tien decaden in de posities 1 t/m 29 zoals hiervoor, de volgende tien decaden in de posities 60 t/m 88. De boodschap SIZE ERROR moet beginnen te verschijnen in de posities 1 of 60, al naar gelang dat vereist is.

4 Gedicht

Een reeks Nederlandse woorden zijn op magneetband onder de label "GEDICHT" opgeslagen. De reeks woorden wordt in regels met een variabele lengte van niet meer dan 80 tekens verdeeld door het enkele teken "/". Ieder fysiek record op de band heeft een lengte van 800 tekens. De laatste regel van een record kan doorlopen op het volgende record, dat wil zeggen een regel kan door de recordgrenzen in tweeën gesplitst worden.

Schrijf een programma dat de band leest en de woorden zodanig afdrukt dat iedere nieuwe regel begint bij positie 30. Het teken "*" duidt aan dat we aan het einde van een regel zijn en dat er geen verdere regels in het record zijn; het volgende record moet worden ingelezen.

De scheidingstekens "/" en "*" moeten niet worden afgedrukt. Lege regels moeten als blanco regels worden afgedrukt.

Aan het einde van de afgedrukte tekst moet u met passende opschriften het aantal klinkers en medeklinkers in het bestand afdrukken (speciale tekens, zoals spaties en cijfers worden niet geteld).

5 Koopgewoontes

(a) Schrijf een programma waardoor een stambestand ontstaat van de koopgewoontes van klanten. Ieder record heeft de volgende vorm:

1-40	Naam en adres
42-45	Rekeningnummer van 4 cijfers
50-55	Aantal geplaatste orders boven ƒ 4.000,-
60-65	Aantal geplaatste orders van ƒ 4.000,- en minder.

De records moeten in volgorde van rekeningnummer staan.

(b) Er bestaan records met de volgende indeling om het bovenstaande bestand bij te werken:

1-4	Rekeningnummer
10-20	Waarde van order deze week geplaatst.

Ga er van uit dat de records op volgorde van rekeningnummer staan en dat de twee velden numeriek zijn. Niet alle records van het stambestand worden noodzakelijkerwijs elke week bijgewerkt. Schrijf een programma voor het bijwerken van het stambestand waarbij een nieuw stambestand wordt opgebouwd, en het afdrukken van iedere mutatie voor niet-bestaande klanten op de regeldrukker. De waarde-velden hebben 2 decimale posities. Ontwerp zelf een indeling voor het foutenverslag.

6 Magisch vierkant

Een magisch vierkant is een twee-dimensionale tabel met de eigenschap dat de som van de getallen in iedere rij, iedere kolom en iedere diagonaal allemaal gelijk zijn.

Schrijf een programma met gebruikmaking van PERFORM... VARYING dat vaststelt of een gegeven 4×4-tabel met tweecijferige getallen wel of geen magisch vierkant is. Druk het vierkant af met een passende boodschap. Voer de te testen vierkanten in op basis van records met een indeling naar eigen keus. (Enthousiastelingen mogen proberen een programma te schrijven voor tabellen van willekeurige omvang tot 20×20 toe, waarbij het record niet alleen de waarde maar ook de dimensie aangeeft.)

7 Aanwezigheid van studenten: deel II

Wijzig opgave 2 zo, dat de regels van het overzicht worden afgedrukt in volgorde van studentnummer binnen volgorde van totaal aantal uren gevolgd binnen volgorde van cursuscode (opklimmend).

8 Koopgewoontes: deel II

Wijzig opgave 5 zo, dat het stambestand een relatief of sequentieel georganiseerd bestand op schijf wordt. Werk de records ter plaatse bij.

9 Kredietkaarthouders

Een bestand van kredietkaarthouders is sequentieel georganiseerd en aangebracht op magneetband. Maandelijks vindt een mutatieverwerking plaats. Het mutatiebestand is eveneens sequentieel georganiseerd en op magneetband aangebracht. Het is mogelijk dat er meerdere mutaties met eenzelfde kaarthoudersnummer bestaan. De mutaties kunnen zijn:
- debetposten, die getotaliseerd moeten worden voor goederen die de houders met hun kaart hebben gekocht,
- creditposten, die in mindering komen op de debettotalen,
- betalingen, die getotaliseerd moeten worden,
- terugbetalingen, die in mindering komen op het betalingentotaal.

De totalen van debetposten en betalingen gedurende een maand worden ook in het stambestand opgenomen. Voordat een record naar het nieuwe stambestand wordt weggeschreven moet het saldo van de houder worden bijgewerkt, dat wil zeggen dat het debettotaal bij het oude saldo moet worden opgeteld, het betalingentotaal ervan moet worden afgetrokken en een rente van 1,5%, te berekenen over het beginsaldo van de maand, erbij moet worden opgeteld. (Het bijgewerkte bestand wordt in een volgende computergang gebruikt om

fakturen aan te maken, waarbij ook de krediet-limiet en het minimum faktuurbedrag worden bepaald, maar dat valt buiten deze opgave.)

Alle mutaties moeten op de regeldrukker worden uitgelijst en ook voor controledoeleinden worden getotaliseerd. Ze moeten namelijk overeenkomen met de aantallen voor de vier soorten mutaties. Deze aantallen staan in het staartlabelrecord waarvan het kaarthoudersnummer de waarde HIGH-VALUE heeft. Bedenk ook controletotalen voor het stambestand-oud, stambestand-nieuw en afdrukbestand.

Als een kaarthoudersnummer van een mutatie niet in het stambestand voorkomt, moet deze mutatie met een foutmelding erbij worden afgedrukt. Treedt een andere fout op, dan moet u een boodschap op het bedieningspaneel afdrukken waaruit blijkt in welke paragraaf deze fout optreedt en de run beëindigen. U dient alles wat u in dit boek hebt geleerd: gestructureerde aanpak, documentatie, controle, enz. in deze opgave toe te passen.

APPENDIX C
LIJST VAN IN ANS-COBOL GERESERVEERDE WOORDEN

ACCEPT
ACCESS
ADD
ADDRESS
ADVANCING
AFTER
ALL
ALPHABETIC
ALSO
ALTER
ALTERNATE
AND
ARE
AREA
AREAS
ASCENDING
ASSIGN
AT
AUTHOR
BEFORE
BLANK
BLOCK
BOTTOM
BY
CALL
CANCEL
CD
CF
CH
CHARACTER
CHARACTERS
CLOCK-UNITS
CLOSE
COBOL
CODE
CODE-SET
COLLATING
COLUMN
COMMA
COMMUNICATION
COMP
COMPUTATIONAL
COMPUTE
CONFIGURATION
CONTAINS
CONTROL
CONTROLS
COPY
CORR
CORRESPONDING
COUNT
CURRENCY
DATA
DATE
DATE-COMPILED
DATE-WRITTEN
DAY
DE
DEBUG-CONTENTS
DEBUG-ITEM
DEBUG-LINE
DEBUG-NAME
DEBUG-SUB-1
DEBUG-SUB-2
DEBUG-SUB-3
DECIMAL-POINT
DECLARATIVES
DELETE
DELIMITED
DELIMITER
DEPENDING
DESCENDING
DESTINATION
DETAIL
DISABLE
DISPLAY
DIVIDE
DIVISION
DOWN
DUPLICATES
DYNAMIC
EGI
ELSE
EMI
ENABLE
END
END-OF-PAGE
ENTER
ENVIRONMENT
EOP
EQUAL
ERROR
ESI
EVERY
EXCEPTION
EXIT
EXTEND
FD
FILE
FILE-CONTROL
FILLER
FINAL
FIRST
FOOTING
FOR
FROM
GENERATE
GIVING
GO
GREATER
GROUP
HEADING
HIGH-VALUE
HIGH-VALUES
I-O
I-O-CONTROL
IDENTIFICATION
IF
IN
INDEX
INDEXED
INDICATE
INITIAL
INITIATE
INPUT
INPUT-OUTPUT
INSPECT
INSTALLATION
INTO
INVALID
IS
JUST
JUSTIFIED
KEY
LABEL
LAST
LEADING
LEFT
LENGTH
LESS
LIMIT
LIMITS
LINAGE
LINAGE-COUNTER
LINE
LINE-COUNTER
LINES
LINKAGE
LOCK
LOW-VALUES
MEMORY
MERGE

MESSAGE
MODE
MODULES
MOVE
MULTIPLE
MULTIPLY
NATIVE
NEGATIVE
NEXT
NO
NOT
NUMBER
NUMERIC
OBJECT-COMPUTER
OCCURS
OF
OFF
OMITTED
ON
OPEN
OPTIONAL
OR
ORGANIZATION
OUTPUT
OVERFLOW
PAGE
PAGE-COUNTER
PERFORM
PF
PH
PIC
PICTURE
PLUS
POINTER
POSITION
POSITIVE
PRINTING
PROCEDURE
PROCEDURES
PROCEED
PROGRAM
PROGRAM-ID
QUEUE
QUOTE
QUOTES
RANDOM
RD
READ
RECEIVE
RECORD
RECORDS
REDEFINES
REEL
REFERENCES
RELATIVE
RELEASE
REMAINDER
REMOVAL
RENAMES
REPLACING
REPORT
REPORTING
REPORTS
RERUN
RESERVE
RESET
RETURN
REVERSED
REWIND
REWRITE
RF
RH
RIGHT
ROUNDED
RUN
SAME
SD
SEARCH
SECTION
SECURITY
SEGMENT-LIMIT
SELECT
SEND
SENTENCE
SEPARATE
SEQUENCE
SEQUENTIAL
SET
SIGN
SIZE
SORT
SORT-MERGE
SOURCE
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAMES
STANDARD
STANDARD-1
START
STATUS
STOP
STRING
SUB-QUEUE-1
SUB-QUEUE-2
SUB-QUEUE-3
SUBTRACT
SUM
SUPPRESS
SYMBOLIC
SYNC
SYNCHRONIZED
TABLE
TALLYING
TAPE
TERMINAL
TERMINATE
TEXT
THAN
THROUGH
THRU
TIME
TIMES
TO
TOP
TRAILING
TYPE
UNIT
UNSTRING
UNTIL
UP
UPON
USAGE
USE
USING
VALUE
VALUES
VARYING
WHEN
WITH
WORDS
WORKING-STORAGE
WRITE
ZERO
ZEROES
ZEROS

APPENDIX D
OVERZICHT VAN FORMATEN VAN ANS-COBOL

Deze vormen weerspiegelen de volledige 1974 American National Standard voor COBOL, uitgezonderd de voorzieningen met betrekking tot berichtenverkeer en uitgezonderd de voorzieningen waarvan CODASYL heeft voorgesteld ze te laten vervallen. In alle gevallen waar THRU staat, is THROUGH een geoorloofd alternatief. De term 'identifier' betekent een normale, een gekwalificeerde of een geïndiceerde gegevensnaam.

OPDRACHT VOOR COMPILER

```
COPY text-name [ { OF } library-name ]
                 { IN }

   [ REPLACING { { <<pseudo-text-1>> }    { <<pseudo-text-2>> } } ]
               { { identifier-1      } BY { identifier-2      } } ] ...
               { { literal-1         }    { literal-2         } } ]
               { { word-1            }    { word-2            } } ]
*

/
```

IDENTIFICATION DIVISION BESCHRIJVINGEN

```
IDENTIFICATION DIVISION.

PROGRAM-ID. program-name.
```

ENVIRONMENT DIVISION BESCHRIJVINGEN

```
ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER. computer-name [WITH DEBUGGING MODE].

OBJECT-COMPUTER. computer-name
```

```
              ⎧WORDS     ⎫
[MEMORY SIZE integer ⎨CHARACTERS⎬]
              ⎩MODULES   ⎭

[PROGRAM COLLATING SEQUENCE is alphabet-name]

[SEGMENT-LIMIT IS segment-number].

SPECIAL-NAMES.

                  ⎧IS mnemonic-name-1   [ON STATUS IS condition-name-1
[implementor-name ⎨IS mnemonic-name-2   [OFF STATUS IS condition-name-2
                  ⎪ON STATUS IS condition-name-5
                  ⎩OFF STATUS IS condition-name-7

                   [OFF STATUS IS condition-name-2]]⎫
                   [ON STATUS IS condition-name-4]] ⎬] ...
                   [OFF STATUS IS condition-name-6] ⎪
                   [ON STATUS IS condition-name-8]  ⎭

                   ⎧STANDARD-1                                       ⎫
                   ⎪NATIVE                                           ⎪
                   ⎪implementor-name                                 ⎪
[alphabet-name IS  ⎨literal-1 [THRU literal-2                    ]   ⎬]
                   ⎪          [ALSO literal-2 [ALSO literal-4]...]   ⎪
                   ⎪[literal-5 [THRU literal-6                    ]] ⎪
                   ⎩           [ALSO literal-7 [ALSO literal-8]...]] ⎭

   CURRENCY SIGN IS literal-9 DECIMAL-POINT IS COMMA .

[INPUT-OUTPUT SECTION.

[FILE-CONTROL.

 (Format 1: Sequential organisation)

  {SELECT [OPTIONAL] file-name ASSIGN TO implementor-name-1

      [implementor-name-2] ...

      [RESERVE integer [AREA ] ORGANIZATION IS SEQUENTIAL
                       [AREAS]

      [ACCESS MODE IS SEQUENTIAL] [FILE STATUS IS data-name]}...

 (Format 2: Relative organisation)

  {SELECT file-name ASSIGN TO implementor-name-1 [implementor-name-2]

      [RESERVE integer [AREA ] ORGANIZATION IS RELATIVE
                       [AREAS]
```

```
[ACCESS MODE IS { SEQUENTIAL [RELATIVE KEY IS data-name-1]  }]
                { {RANDOM } RELATIVE KEY IS data-name-1      }
                { {DYNAMIC}                                  }

[FILE STATUS IS data-name-2]} ...
```

(Format 3: Indexed organisation)

```
{SELECT file-name ASSIGN TO implementor-name-1 [implementor-name-2]...

   [RESERVE integer [AREA ] ORGANIZATION IS INDEXED
                    [AREAS]

   [ACCESS MODE IS {SEQUENTIAL}] RECORD KEY IS data-name-1
                   {RANDOM    }
                   {DYNAMIC   }

   [ALTERNATE RECORD KEY IS data-name-2 [WITH DUPLICATES]] ...

   [FILE STATUS IS data-name-3]}...]
```

```
[I-O-CONTROL.

   [RERUN [ON {file-name-1        }]
              {implementor-name   }

   EVERY {{END OF {UNIT}        }   OF file-name-2 }]
         {{        {REEL}       }                  }
         {{integer-1 RECORDS    }                  }
         {integer-2 CLOCK-UNITS                    }
         {condition-name                           }

   [SAME [RECORD    ] AREA FOR file-name-3 [file-name-4]...]...
         [SORT      ]
         [SORT-MERGE]

   [MULTIPLE FILE TAPE CONTAINS file-name-5 [POSITION integer-3]

      [file-name-6 [POSITION integer-4]]...]...   .]]
```

DATA DIVISION BESCHRIJVINGEN

```
[FILE SECTION.

[FD file-name

   [BLOCK CONTAINS [integer-1 TO] integer-2 {RECORDS   }]
                                            {CHARACTERS}

   [RECORD CONTAINS [integer-3 TO] integer-4 CHARACTERS]

   LABEL {RECORDS ARE} {STANDARD}
         {RECORD IS  } {OMITTED }
```

```
    [VALUE OF implementor-name-1 IS {data-name-1} ...]
                                    {literal-1  }

    [LINAGE IS {data-name-5} LINES [WITH FOOTING AT {data-name-6}]
               {integer-5  }                        {literal-6  }

    [LINES AT TOP {data-name-7}][LINES AT BOTTOM {data-name-8}]
                  {integer-7  }                  {integer-8  }

    [CODE-SET IS alphabet-name]

   [{REPORT IS  } report-name-1 [report-name-2]...].
    {REPORTS ARE}

[record-description-entry]...]...

[WORKING-STORAGE SECTION.

[77-level-description-entry]...]
[record-description-entry  ]

[LINKAGE SECTION.

[77-level-description-entry]...]
[record-description-entry  ]

[REPORT SECTION.

[RD report-name

   [CODE literal-1]

  [{CONTROLS ARE} {data-name-1 [data-name-2]...              }]
  [{CONTROL IS  } {FINAL [data-name-1 [data-name-2]...]      }]

   [PAGE [LIMIT IS  ] integer-1 [LINE ] HEADING integer-2]
         [LIMITS ARE]           [LINES]

   [FIRST DETAIL integer-3] [LAST DETAIL integer-4]

   [FOOTING integer-5]].

{report-group-description-entry}...]...]
```

BESCHRIJVINGEN VAN GEGEVENS

(Format 1)

```
level-number {data-name-1} [REDEFINES data-name-2]
             {FILLER     }

  [{PICTURE} IS picture-string]
   {PIC    }
```

```
          ⎧ COMPUTATIONAL ⎫
[USAGE IS ⎨ COMP          ⎬ ]
          ⎪ DISPLAY       ⎪
          ⎩ INDEX         ⎭

[[SIGN IS] { LEADING  } [SEPARATE CHARACTER]]
           { TRAILING }

[OCCURS { integer-1 TO integer-2 TIMES DEPENDING ON data-name-3 }
        { integer-2 TIMES                                       }

   [{ ASCENDING  } KEY IS data-name-4 [data-name-5]...]...
    { DESCENDING }

   [INDEXED BY index-name-1 [index-name-2]...]]

[{ SYNCHRONIZED } [ LEFT  ]]
 { SYNC         } [ RIGHT ]

[{ JUSTIFIED }  RIGHT]
 { JUST      }

[BLANK WHEN ZERO]

[VALUE IS literal].
```

(Format 2)

```
66 data-name-1 RENAMES data-name-2  [THRU data-name-3]
```

(Format 3)

```
88 condition-name { VALUE IS   } literal-1 [THRU literal-2]
                  { VALUES ARE }
                                    [literal-3 [THRU literal-4]]...
```

BESCHRIJVINGEN VAN REPORT GROUP

(Format 1)

```
01 [data-name-1]

   [LINE NUMBER IS { integer-1 on NEXT PAGE }]
                   { PLUS integer-2         }

                   ⎧ integer-3      ⎫
   [NEXT GROUP IS  ⎨ PLUS integer-4 ⎬]
                   ⎩ NEXT PAGE      ⎭
```

```
             ⎧REPORT HEADING                    ⎫
             ⎪RH                                ⎪
             ⎪PAGE HEADING                      ⎪
             ⎪PH                                ⎪
             ⎪CONTROL HEADING data-name-2       ⎪
             ⎪CH              FINAL             ⎪
    TYPE IS  ⎨DETAIL                            ⎬
             ⎪DE                                ⎪
             ⎪CONTROL FOOTING data-name-3       ⎪
             ⎪CF                                ⎪
             ⎪PAGE FOOTING                      ⎪
             ⎪PF                                ⎪
             ⎪REPORT FOOTING                    ⎪
             ⎩RF                                ⎭

     [[USAGE IS] DISPLAY].
```

(Format 2)

```
level-number [data-name-1]

    LINE NUMBER IS {integer-1 ON NEXT PAGE}
                   {PLUS integer-2        }

  [[USAGE IS] DISPLAY].
```

(Format 3)

```
level-number [data-name-1]

   [BLANK WHEN ZERO]

   [GROUP INDICATE]

  [{JUSTIFIED}  RIGHT]
   {JUST     }

   [LINE NUMBER IS {integer-1 ON NEXT PAGE} ]
                   {PLUS integer-2        }

   [COLUMN NUMBER IS integer-3]

   {PICTURE}  IS picture-string
   {PIC    }

   ⎧SOURCE IS identifier-1                                   ⎫
   ⎪VALUE IS literal                                         ⎪
   ⎨{SUM identifier-2 [identifier-3]...                      ⎬
   ⎪      [UPON data-name-2 [data-name-3]...]}...            ⎪
   ⎪           [RESET ON {data-name-4}]                       ⎪
   ⎩                     {FINAL      }                        ⎭

  [[USAGE IS] DISPLAY].
```

PROCEDURE DIVISION BESCHRIJVINGEN

```
PROCEDURE DIVISION [USING data-name-1 [data-name-2]...]

(Format 1 with sections)

{section-name SECTION [segment-number].

[paragraph-name. [sentence]...]...}...

(Format 2 without sections)

{paragraph-name. [sentence]...}...

(Format 3 with declaratives)

DECLARATIVES.

{declarative-section}...

END DECLARATIVES.

{section-name SECTION [segment-number].

[paragraph-name. [sentence]...]...}...
```

COBOL-OPDRACHTEN

```
                             { mnemonic-name }
ACCEPT identifier [FROM      { DATE          } ]
                             { DAY           }
                             { TIME          }

ADD {identifier-1} [identifier-2] TO identifier-m  [ROUNDED]
    {literal-1   } [literal-2   ]

   [identifier-n [ROUNDED]]... [ON SIZE ERROR imperative-statement]

ADD {identifier-1} {identifier-2} [identifier-3] ... GIVING
    {literal-1   } {literal-2   } [literal-3   ]

    identifier-m [ROUNDED] [ON SIZE ERROR imperative-statement]

CALL {identifier-1} [USING data-name-1 [data-name-2]...]
     {literal-1   }

     [ON OVERFLOW imperative-statement]

CANCEL {identifier-1} [identifier-2]...
       {literal-1   } [literal-2   ]
```

```
                     [{REEL} [WITH NO REWIND]]
                     [{UNIT} [FOR REMOVAL   ]]
CLOSE file-name-1    [                        ]
                     [WITH {NO REWIND}        ]
                     [     {LOCK     }        ]

                     [{REEL} [WITH NO REWIND]]
                     [{UNIT} [FOR REMOVAL   ]]
      [file-name-2   [                        ] ]
                     [WITH {NO REWIND}        ]
                     [     {LOCK     }        ]

COMPUTE identifier-1 [ROUNDED][identifier-2 [ROUNDED]]...

      {identifier-2         }
    = {literal              } [ON SIZE ERROR imperative-statement]
      {arithmetic expression}

DELETE file-name RECORD [INVALID KEY imperative-statement]

DISPLAY {identifier-1} [identifier-2] ... [UPON mnemonic-name]
        {literal-1   } [literal-2   ]

DIVIDE {identifier-1}  INTO identifier-2 [ROUNDED]
       {literal-1   }

       [identifier-3 [ROUNDED]]...[ON SIZE ERROR imperative-statement]

DIVIDE {identifier-1} {BY  } {identifier-2}  GIVING identifier-3
       {literal-1   } {INTO} {literal-2   }

       [ROUNDED] [identifier-4 [ROUNDED]] ...

       [ON SIZE ERROR imperative-statement]

DIVIDE {identifier-1} {BY  } {identifier-2}  GIVING identifier-3
       {literal-1   } {INTO} {literal-2   }

       [ROUNDED] REMAINDER identifier-4

       [ON SIZE ERROR imperative-statement]

EXIT [PROGRAM].

GENERATE {data-name  }
         {report-name}

GO TO procedure-name

GO TO procedure-name-1 [procedure-name-2]... DEPENDING ON identifier

IF condition {statement-1  } [ELSE statement-2   ]
             {NEXT SENTENCE} [ELSE NEXT SENTENCE ]
```

```
INSPECT identifier-1 TALLYING

   {identifier-2 FOR {{ {ALL    } {identifier-3} }
                     {{ {LEADING} {literal-1   } }
                     {{ CHARACTERS               }

        { {BEFORE}  INITIAL  {identifier-4} } } }...}...
        { {AFTER }           {literal-2   } } } }

INSPECT identifier-1 REPLACING

  {  CHARACTERS BY {identifier-6} [{BEFORE} INITIAL {identifier-7}]
  {                {literal-4   } [{AFTER }         {literal-5   }]
  {
  { { {ALL    } { {identifier-5} BY {identifier-6}
  { { {LEADING} { {literal-3   }    {literal 5   }
  { { {FIRST  }
  {
  {    [{BEFORE} INITIAL  {identifier-7}]}...}...
  {    [{AFTER }          {literal-5   }]}

INSPECT identifier-1 TALLYING as above REPLACING as above

MERGE file-name-1 ON {ASCENDING } KEY data-name-1 [data-name-2]...
                     {DESCENDING}

              [ON {ASCENDING } KEY data-name-3 [data-name-4]...]...
                  {DESCENDING}

   [COLLATING SEQUENCE IS alphabet-name]

   USING file-name-1  file-name-2 [file-name-3]...

   {OUTPUT PROCEDURE   IS section-name-1 [THRU section-name-2]}
   {GIVING file-name-5                                        }

MOVE {identifier-1} TO identifier-2 [identifier-3]...
     {literal-1   }

MULTIPLY {identifier-1} BY identifier-2 [ROUNDED]
         {literal-1   }

   [identifier-3 [ROUNDED]]...

   [ON SIZE ERROR imperative-statement]

MULTIPLY {identifier-1} BY {identifier-2} GIVING identifier-3
         {literal-1   }    {literal-2   }

   [ROUNDED] [identifier-4 [ROUNDED]]...

   [ON SIZE ERROR imperative-statement]
```

```
       ⎧ INPUT file-name-1 [WITH NO REWIND][file-name-2[WITH NO REWIND]]... ⎫
OPEN   ⎨ OUTPUT file-name-3[WITH NO REWIND][file-name-3[WITH NO REWIND]]... ⎬
       ⎪ I-O file-name-5 file-name-6 ...                                    ⎪
       ⎩ EXTEND file-name-7  file-name-8 ...                                ⎭

       ⎧ INPUT file-name-1 [file-name-2]...  ⎫
OPEN   ⎨ OUTPUT file-name-3 [file-name-4]... ⎬...
       ⎩ I-O file-name-5 [file-name-6]...    ⎭

PERFORM procedure-name-1 [THRU procedure-name-1]

       ⎡ ⎧identifier-1⎫       ⎤
       ⎢ ⎩integer     ⎭ TIMES ⎥
       ⎣ UNTIL condition      ⎦

PERFORM procedure-name-1 [THRU procedure-name-2]

    VARYING ⎧index-name-1⎫ FROM ⎧index-name-2⎫ BY ⎧literal-2   ⎫
            ⎩identifier-1⎭      ⎨literal-1   ⎬    ⎩identifier-3⎭
                                ⎩identifier-2⎭

         UNTIL condition-1

[AFTER ⎧index-name-3⎫ FROM ⎧index-name-4⎫ BY ⎧literal-4   ⎫
       ⎩identifier-4⎭      ⎨literal-3   ⎬    ⎩identifier-6⎭
                           ⎩identifier-5⎭

         UNTIL condition-2

[AFTER ⎧index-name-5⎫ FROM ⎧index-name-6⎫ BY ⎧literal-6   ⎫
       ⎩identifier-7⎭      ⎨literal-5   ⎬    ⎩identifier-9⎭
                           ⎩identifier-8⎭

         UNTIL condition-3]]

READ file-name [NEXT] RECORD [INTO identifier]

   [AT END imperative-statement]

READ file-name RECORD [INTO identifier] [KEY IS data-name]

   [INVALID KEY imperative-statement]

RELEASE record-name [FROM identifier]

RETURN file-name record [INTO identifier]

    AT END imperative-statement

REWRITE record-name [FROM identifier]

   [INVALID KEY imperative-statement]
```

```
SEARCH identifier-1 [VARYING { index-name   }]
                             { identifier-2 }

   [AT END imperative-statement-1]

   WHEN condition-1 { imperative-statement-2 }
                    { NEXT SENTENCE          }

   [WHEN condition-2 { imperative-statement-3 }]...
                     { NEXT SENTENCE          }

SEARCH ALL identifier-1

   [AT END imperative-statement-1]

   WHEN special-condition-1 [AND special-condition-2]...

       { imperative-statement-2 }
       { NEXT SENTENCE          }

(special condition)

  { data-name-1 { IS EQUAL TO } { identifier-3          } }
  {             { IS =        } { literal-1             } }
  {                             { arithmetic-expression } }
  { condition-name-1                                      }

SET { index-name-1 } { [index-name-2]... } TO { index-name-3 }
    { identifier-1 } { [identifier-2]... }    { literal-1    }
                                              { integer      }

SET  index-name-1  [index-name-2]... { UP BY   } { identifier }
                                     { DOWN BY } { integer    }

SORT file-name-1 ON { ASCENDING  } KEY  identifier-1 ...
                    { DESCENDING }

   [ON { DESCENDING } KEY  identifier-2 ...]...
       { ASCENDING  }

   [COLLATING SEQUENCE IS alphabet-name]

   { INPUT PROCEDURE IS section-name [THRU section-name-2] }
   { USING file-name-2  file-name-3 ...                    }

   { OUTPUT PROCEDURE IS section-name-3 [THRU section-name-4] }
   { GIVING file-name-3                                       }

                               { EQUAL TO       }
                               { =              }
START file-name [KEY IS        { GREATER THAN   }  data-name]
                               { >              }
                               { NOT LESS THAN  }
                               { NOT <          }

   [INVALID KEY imperative-statement]
```

```
STOP {literal}
     {RUN    }

STRING {identifier-1} [identifier-2]... DELIMITED BY {identifier-3}
       {literal-1   } [literal-2   ]                 {literal-3   }
                                                     {SIZE        }

    [{identifier-4} [identifier-5]... DELIMITED BY {identifier-6}]...
    [{literal-4   } [literal-5   ]                 {literal-6   }]
                                                   {SIZE        }
   INTO identifier-7 [WITH POINTER identifier-8]

  [ON OVERFLOW imperative-statement]

SUBTRACT {literal-1   } [literal-2   ]... FROM identifier-m
         {identifier-1} [identifier-2]

  [ROUNDED] [identifier-n [ROUNDED]]...

  [ON SIZE ERROR imperative-statement]

SUBTRACT {literal-1   } [literal-2   ] ... FROM {literal-m   }
         {identifier-1} [identifier-2]          {identifier-m}

   GIVING identifier-n [ROUNDED]...[identifier-o [ROUNDED]]...

  [ON SIZE ERROR imperative-statement]

SUPPRESS PRINTING

UNSTRING identifier-1 [DELIMITED BY [ALL] {identifier-2}
                                          {literal-1   }

                               [OR [ALL] {identifier-3}]...]
                                         {literal-2   }
   INTO identifier-4 [DELIMITER IN identifier-5]

        [COUNT IN identifier-6] [identifier-7

        [DELIMITER IN identifier-8] [COUNT IN identifier-9]]...

  [WITH POINTER identifier-10] [TALLYING IN identifier-11]

  [ON OVERFLOW imperative-statement]
```

```
USE AFTER STANDARD {EXCEPTION} PROCEDURE ON
                   {ERROR    }

    {file-name-1  file-name-2}
    {INPUT                   }
    {OUTPUT                  }
    {I-O                     }
    {EXTEND                  }

USE BEFORE REPORTING identifier

WRITE record-name [FROM identifier]

    [INVALID KEY imperative-statement]

WRITE record-name [FROM identifier-1]

                          {identifier LINES}
    {BEFORE} ADVANCING    {integer LINES   }
    {AFTER }              {mnemonic-name   }
                          {PAGE            }

    [AT {END-OF-PAGE} imperative-statement]
        {EOP        }
```

CONDITIES

```
                                           {GREATER THAN}
{identifier-1             }                {LESS THAN   }  {identifier-2        }
{literal-1                }                {EQUAL TO    }  {literal-2           }
{arithmetic-expression-1  }  IS [NOT]      {>           }  {arith-expression-2  }
{index-name-1             }                {<           }  {index-name-2        }
                                           {=           }

  identifier IS [NOT] {NUMERIC     }
                      {ALPHABETIC  }

                                      {POSITIVE}
  arithmetic expression is [NOT]      {NEGATIVE}
                                      {ZERO    }
```

INDEX

ACADEMIC SERVICE INFORMATICA UITGAVEN

AUTOMATISERING EN COMPUTERS

Computers en onze informatiemaatschappij - M.A. Arbib
Computers in de negentiger jaren - G.L. Simons
De informatiemaatschappij - Jan Everink
Op weg naar een risicoloze maatschappij? - Jan Holvast
Basiskennis informatieverwerking - Jan Everink
AIV, Automatisering van de informatieverzorging - Th.J.G. Derksen & H.W. Crins
BIV, Basis van de geautomatiseerde informatieverzorging - Th.J.G. Derksen & H.W. Crins
Organisatie, informatie en computers - David M. Kroenke
Computers in de basisschool - H. Lamers & J.A. Wegkamp
Effectieve toepassingen van computers - M. Peltu

MICROCOMPUTERS

Microcomputers thuis en op school - K.P. Goldberg & D. Sherwood
Bouw zelf een expertsysteem in BASIC - Chris Naylor
Programmeercursus MicrosoftBASIC - Nok van Veen
Werken met bestanden in BASIC - LeRoy Finkel & Jerald R. Brown
Programmeercursus BASIC op de Commodore 64 - Nok van Veen
Werken met bestanden op de Commodore 64 - G. Fisher, L. Finkel & J.R. Brown
Het Electron en BBC Micro boek - Jim McGregor & Alan Watt
Werken met bestanden op de Apple - LeRoy Finkel & Jerald R. Brown
Programmeercursus ApplesoftBASIC - Nok van Veen & Ad van Delft
Programmeercursus MSX BASIC - Nok van Veen
Werken met bestanden in MSX BASIC - LeRoy Finkel & Jerald R. Brown
40 grafische programma's - voor de Commodore 64; voor de Electron en BBC; voor de ZX Spectrum; voor de Apple II, IIe en IIc; in MSX BASIC - Marcel Sutter

MICROPROCESSORS EN ASSEMBLEERTALEN

Procescomputers, basisbegrippen - J.E. Rooda & W.C. Boot
Cursus Z-80 assembleertaal - Roger Hutty
6502 assembleertaal en machinecode voor beginners - A.P. Stephenson
EXAT-handboek - Micro-Teach

BESTURINGSSYSTEMEN

Inleiding besturingssystemen - A.M. Lister
Theorie en praktijk van besturingssystemen - J.L. Peterson & A. Silberschatz
Systeemprogrammatuur en softwareontwikkeling voor microcomputers - E. Verhulst
Bedrijfssystemen - EIT-serie, deel 4
CP/M, het operating system voor microcomputers - J.N. Fernandez & R. Ashley
CP/M 86, een besturingssysteem voor 16 bit microcomputers - J.N. Fernandez & R. Ashley
CP/M voor gevorderden - A. Clarke e.a.
PC DOS, het besturingssysteem van de IBM PC - R. Ashley & J.N. Fernandez
MS DOS, het besturingssysteem voor 16 bit microcomputers - R. Ashley & J.N. Fernandez
UNIX, het standaard operating system - G.J.M. Austen & H.J. Thomassen
De UNIX programmeeromgeving - B.W. Kernighan & R. Pike

PERSONAL COMPUTERS EN PROGRAMMAPAKKETTEN

De IBM PC en zijn toepassingen - Laurence Press
Werken met bestanden in IBM- en GW-BASIC - J.R. Brown & LeRoy Finkel
40 grafische programma's in IBM- en GW-BASIC - Marcel Sutter
Werken met VisiCalc - C. Klitzner & M.J. Plociak Jr.
Multiplan, een hulpmiddel bij de bedrijfsvoering - D.F. Cobb e.a.
Werken met Lotus 1-2-3 - G.T. LeBlond & D.F. Cobb
Lotus 1-2-3: Tips, Trucs en Tegenvallers - D. Andersen & D.F. Cobb
Lotus 1-2-3: Financiële macro's - Thomas W. Carlton
Symphony deel I en II - D.P. Ewing & G.T. LeBlond
dBASE III handboek - George Tsu-der Chou
WordStar stap voor stap - Ruth Ashley & Judi N. Fernandez

PROGRAMMEREN

Een methode van programmeren - Edsger W. Dijkstra & W.H.J. Feijen
Programmeren, met ontwerpen van algoritmen (met Pascal) - J.J. van Amstel
Voortgezet programmeren, het ontwerpen van datastructuren en algoritmen - J.J. van Amstel & J.A.A.M. Poirters

Over dit boek

Het eerste deel van dit boek behandelt die begrippen die de student in staat stellen eenvoudige COBOL-programma's te schrijven, die gebruik maken van sequentieel georganiseerde bestanden. Omdat gestructureerd programmeren inmiddels dermate is ingeburgerd zijn in de tweede hierziene versie van dit succesvolle boek in het tweede deel enige hoofdstukken over dit onderwerp toegevoegd. Hierin wordt behandeld hoe de analyse van gegevensstructuren resulteert in een gestructureerd programmaontwerp en hoe dit ontwerp in COBOL kan worden weergegeven. Een andere belangrijke toevoeging betreft het programmeren van interactieve dialogen. Het boek is nu volledig gebaseerd op de laatste (1974) ANS standaard en belangrijke voorstellen voor wijzigingen van de ANS COBOL-standaard die de afgelopen jaren zijn ingediend worden waar toepasselijk vermeld. In het derde gedeelte van het boek wordt de nodige aandacht besteed aan o.a.:

- het werken met direct georganiseerde bestanden;
- het werken met de report writer;
- sorteren.